UCLA
UCLA Electronic Theses and Dissertations

Title

Integrating Language Models with Evolving External Knowledge

Permalink

https://escholarship.org/uc/item/3gj3t6ct

ISBN

9798173781741

Author

Wu, Di

Publication Date

2026-11-10

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA

Los Angeles

Integrating Language Models with Evolving External Knowledge

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Di Wu

2026

© Copyright by
Di Wu
2026

ABSTRACT OF THE DISSERTATION

Integrating Language Models with Evolving External Knowledge

by

Di Wu
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2026
Professor Kai-Wei Chang, Chair

Large language models acquire broad capabilities from public training data, but their deployment increasingly takes place in private worlds shaped by individual users, organizations, and specialized environments. These worlds contain personally and economically valuable knowledge that is local, dynamic, and accumulated through continued interaction and experience. This dissertation studies how language models can effectively integrate such evolving external knowledge. We first characterize how the knowledge requirements of deployed systems expand as applications progress from chat assistants to long-horizon agents. To address these challenges, we develop a two-sided approach to knowledge augmentation. On the model side, we study how language models can recognize the boundary of their own knowledge and selectively draw from multiple knowledge sources. On the knowledge side, we design increasingly expressive representations that make it possible to discover, organize, and reuse broader forms of domain, reasoning, and organizational knowledge. Together, these directions provide a unified framework for language models to continually acquire and use external knowledge as they adapt to evolving worlds.

The dissertation of Di Wu is approved.

Saadia Gabriel

Yizhou Sun

Guy Van den Broeck

Kai-Wei Chang, Committee Chair

University of California, Los Angeles

2026

TABLE OF CONTENTS

1 Introduction 1

I Understanding Knowledge Challenges in Private Worlds 8

2 Long-Term Memory for Personal Chat Assistants 9

2.1 Related Work 11

2.2 LongMemEval 14

2.3 A Unified View of Long-Term Memory Assistants 21

2.4 Experiment Results 24

2.5 Conclusion 29

3 Memory for Experienced Agents in Specialized Environments 31

3.1 Related Work 35

3.2 LongMemEval-V2 36

3.3 AgentRunbook 43

3.4 Experiments 46

3.5 Conclusion 52

3.6 Limitations 52

II Selective Knowledge Access 54

4 Selective Retrieval for Repository-Level Code Completion 55

4.1 Related Work 58

4.2 Approach . . . 59
4.3 Experimental Setup . . . 64
4.4 Results . . . 67
4.5 Analysis . . . 73
4.6 Conclusion . . . 79

5 Routing Between Parametric and External Knowledge . . . 80
5.1 Related Work . . . 83
5.2 Approach . . . 84
5.3 Experimental Setup . . . 90
5.4 Results . . . 92
5.5 SR-RAG: Multi-Source Extensions . . . 99
5.6 Conclusion . . . 101

III Framing, Discovering, and Evolving Reusable Knowledge . . . 103

6 Discovering Domain-Specific Policy Knowledge . . . 104
6.1 Related Work . . . 106
6.2 Approach . . . 107
6.3 Results: OpenAI Content Moderation . . . 109
6.4 Results: In-house Multimodal Advertisement Moderation . . . 111
6.5 Further Analyses . . . 114
6.6 Conclusion . . . 114

7 Building a General Procedural Reasoning Datastore 118

7.1 Related Work 121

7.2 Reasoning Memory: Approach 123

7.3 Experimental Setup 127

7.4 Results 129

7.5 Conclusion 140

7.6 Limitations and Reproducibility 141

8 Evolving Organizational Reflection Skills for Digital Agents 142

8.1 Related Work 145

8.2 Organizational Reflection Layer 146

8.3 Continually Learning Organizational Reflection Skills 148

8.4 Evaluation Setup 152

8.5 Experiments and Analysis 157

8.6 Conclusion 165

9 Conclusion and Future Directions 166

9.1 Conclusions 166

9.2 Future Directions 167

References 169

LIST OF FIGURES

2.1 Examples of the seven question types in LongMemEval. Each example shows the associated evidence statements on the left and the question with the answer on the right. 13

2.2 Data creation pipeline of LongMemEval. (a) Human experts construct questions and evidence statements. (b) The evidence sessions are LLM-simulated and human-edited. (c) At test time, we construct the user-AI history with freely configurable length. 17

2.3 LongMemEval challenges chat assistants through its (a) diverse question types, (b) emphases on multi-session reasoning, and (c) diverse evidence locations within sessions. 18

2.4 Pilot study of (a) commercial systems and (b) long-context LLMs. 20

2.5 A unified view of a chat assistant with long-term memory in operation. We formulate three stages and four control points (CP). We provide further examples in Table 2.3 and Appendix C of the original paper [WWY25]. 22

2.6 QA performance on $\text{LongMemEval}_{\text{M}}$ with different value designs. Decomposing sessions into rounds improves the QA performance. For the multi-session reasoning questions, further representing the values with the extracted facts improves the QA accuracy. 24

2.7 Question answering performance under the oracle retrieval setting. CoN with JSON format outperforms the other three parameter combinations by a large margin. NL = Natural Language. 28

2.8 An analysis of the error distribution of different reader models. We use R and G to indicate correct Recall@10 and correct answer based on the top-10 retrieved results. 29

3.1 Examples of LongMemEval-V2 questions. We display examples from the WorkArena domain. LME-V2 questions exercise diverse memory abilities: static (row 1), dynamic (row 2), workflow (row 3), gotchas (row 4), and premise awareness (right column). 33

3.2 LME-V2 questions cover diverse source domains, question types, and formats. . 38

3.3 Basic statistics of history haystacks in LME-V2. The small and medium tiers cover a large number of states and tokens (left). Meanwhile, the trajectories containing answers to each question remain generally sparse in the haystack (right). 39

3.4 Pilot studies on LME-V2 non-abstention questions. (a) Frontier LLMs perform poorly without trajectory history, suggesting that parametric knowledge alone is insufficient for LME-V2. (b) LME-V2 is challenging to answer even with oracle answer-bearing trajectories and optimizations such as evidence slicing with notes or using a coding agent harness help improve performance. 42

3.5 An illustration of the AgentRunbook memory modules. (a) AgentRunbook-R digests trajectories at insertion time and creates dedicated knowledge pools for raw states, events, and notes. At query time, an LLM controller generates multi-stream queries for each pool. (b) AgentRunbook-C stores trajectories as files at insertion time and constructs a sandbox workspace with specialized instructions and manifests for a coding agent to collect the required evidence efficiently. . . . 44

3.6 AgentRunbook improves the query accuracy-latency frontier of memory modules. The panels compare LME-V2-Small and LME-V2-Medium. Each operating point shows answer accuracy and query latency under the labeled reasoning effort setting. AgentRunbook-R supports efficient querying, while AgentRunbook-C improves the trade-off over the Codex baseline. 48

3.7 Error composition for the main methods on LME-V2-Small and LME-V2-Medium. Each horizontal bar decomposes the incorrect answers for one method into retrieval misses, reading failures, gotcha failures, and premise-awareness failures. 51

3.8 Mean number of command executions per query divided by command types. Each stacked bar compares the command use of AgentRunbook-C and Codex on LME-V2-Small or LME-V2-Medium. AgentRunbook-C reduces the total number of commands and shifts work from raw trajectory exploration toward harness-guided retrieval. 52

4.1 An overview of the proposed selective RAG framework. Given the current file context, the system first assesses whether retrieval is required and triggers the retriever if the question can likely be benefited from retrieval (right), abstaining from retrieval otherwise (left). Then, the code LM generates with optional retrieved contexts. With REPOFORMER, the two stages are streamlined via self-assessment. 55

4.2 A comparison between fill-in-the-middle and self-selective RAG. We mark the end of the current file with a new token `<eof>`, which triggers the LM's self-evaluation. $\rightarrow$ denotes the invocation of the LM. We color current-file context, retrieved contexts, and LM-generated parts in blue, green, and red respectively. `fim_p`, `fim_s`, and `fim_m` refer to the special tokens for fill-in-the-middle: `fim_prefix`, `fim_suffix`, and `fim_middle`. These tokens are already learned during the pre-training. 63

4.3 The performance gain on RepoEval API completion from retrieved cross-file contexts. Each bucket contains values ranging from label-10 to label+10 except for the central bucket, which corresponds to exactly 0. The retrieved contexts only improve the performance in about 20% of instances. The trend is consistent across all the evaluated LM families and sizes. 65

4.4 The accuracy and latency change with different threshold settings. Selective Retrieval with REPOFORMER achieves better accuracy and better latency than always retrieving. In addition, this behavior is relatively insensitive to the threshold. 74

4.5 An analysis of the instances where REPOFORMER-1B abstains from retrieval. We divide the instances into (1) the model answering correctly without retrieval (dark blue), the model making a mistake that cannot be improved by retrieval (light blue), and the model achieving better performance when retrieval is performed (red). The precision of abstention is over 0.8 on all tasks except for Function (RepoEval), which has a precision of 0.78. 75

4.6 The performance change on RepoEval from retrieved cross-file context for the instances where REPOFORMER self-selects retrieval. Compared to StarCoderBase, REPOFORMER is better at leveraging CC to improve the generation quality. . . 76

5.1 **Self-Routing RAG Inference.** (1) A knowledge source is selected based on a query by combining the LLM's likelihood of a special source token with neighbor decisions from an external policy datastore. (2) If an external source is selected, the passages are retrieved and appended as context. If the internal source is selected, the LLM generates a compact verbalized knowledge context. (3) The LLM then produces the final answer conditioned on the query and the selected knowledge. All steps can be executed in a single left-to-right generation pass. . . 81

5.2 **SR-RAG expands selective retrieval into source routing.** Compared to traditional selective retrieval, SR-RAG allows the LLM to route among knowledge sources and to act as a knowledge source itself. Blue denotes external retrieved content and red denotes the LLM and its self-generated tokens. 86

5.3 **Knowledge verbalization significantly affects the LLM ability boundary.** For a large number of instances (16.4% - 38.8%, orange), GenRead reverses the knowledge source preferences: without considering GenRead, RAG dominates over parametric knowledge. 92

5.4 **Accuracy and system latency of SR-RAG.** We show the performance of fine-tuned Llama-2-7B-Chat with different verbalization frequencies. SR-RAG's source selection policy (marked with stars) achieves near-optimal accuracy-efficiency trade-off without dataset-specific thresholds. 95

6.1 An illustration of Deep Policy Research. Based on a domain specification, an LLM iteratively interacts with a search engine, extracts policy rules, and indexes the rules through keyphrase-based clustering. 105

6.2 Rule count and cluster count vs. iteration in the Harassment domain for DPR initialized with two LLMs. The left panel uses Llama 3.3 70B as the research model and the right uses Qwen2.5 32B. Blue circles show rule counts on the left axis, and orange squares show keyphrase-cluster counts on the right axis. 115

6.3 Performance vs. iteration in the Harassment domain with two different LLMs for DPR and Qwen2.5 7B Instruct as the reader. The panels use Llama 3.3 70B and Qwen2.5 32B as the research models. We compare moderation F_1 using a flat rule list, clustered rules, and cluster summaries. The dotted line marks the zero-shot baseline. 115

6.4 Performance vs. iteration in the Harassment domain with two different LLMs for DPR and Llama 3.1 8B Instruct as the reader. The panels use Llama 3.3 70B and Qwen2.5 32B as the research models. We compare moderation F_1 using a flat rule list, clustered rules, and cluster summaries. The dotted line marks the zero-shot baseline. 116

7.1 **Illustration of the Reasoning Memory framework.** (a) We extract self-contained procedural knowledge from diverse public reasoning trajectories to construct a datastore. (b) At inference time, we retrieve top-k subquestion-subroutine pairs, generate separate groups of candidate trajectories under each subroutine, and filter and rank the candidates. The source trajectories are not inserted. 120

7.2 **Standard document RAG benefits instruction-tuned models more than reasoning models.** Under the CompactDS pipeline, instruction-tuned models obtain modest gains from retrieval, whereas the corresponding reasoning models often see limited gains or even degradation, despite much stronger no-retrieval performance. 130

7.3 **Utility of different types of synthesized knowledge.** We report performance gains relative to the no-retrieval setting. Both factual and procedural knowledge help, but procedural knowledge yields larger gains across tasks and model families. 131

7.4 **Performance as a function of inference budget.** We compare Length Scaling without retrieval with two Reasoning Memory variants on DeepSeek-R1-Distill-Llama-8B as the total sampling budget m increases. Intensity-First allocates 20 samples to each selected subroutine, while Diversity-First distributes the budget across the top 20 retrieved subroutines. The panels report accuracy on AIME 2024, GPQA-D, and LiveCodeBench. 134

7.5 **Effect of datastore size and composition on Reasoning Memory.** Performance of DeepSeek-R1-Distill-Llama-8B with budget $m = 30$. Larger and more diverse datastores generally yield stronger performance. The connected curve uses random subsets spanning 10% to 100% of Nemotron trajectories. Separate markers show math-only, code-only, and science-only Nemotron subsets and an OpenThoughts3 datastore. 135

8.1 **Organizational reflection around a single worker.** (a) ReOrg consults a virtual organization of simulated stakeholders with separate read-only contexts. An assembled team produces a handoff that guides repeated revisions by the same task-owning worker. (b) GDPval win+tie rates compare a single worker, one round of self-reflection, and ReOrg with up to three reflection rounds. . . . 143

8.2 **Continual learning of reflection skills and retrieval knowledge.** (a) For each unlabeled task, ReOrg retrieves and evolves skills, then promotes useful structures and summarizes outcomes into retrieval knowledge for subsequent tasks. (b) A label-free judge compares blinded artifacts produced after reflection and worker revision. The resulting ranks guide skill selection, further mutation, and final promotion. 149

8.3 Pilot study on GDPval. (a) Win + tie rates for Gemini 3.1 Pro under four reflection conditions, shown on a 40.0–60.0% scale. (b) Win + tie rates with one or three organizational reflection rounds, normalized by each model's single-worker rate (100.0%). 158

8.4 Cross-domain transfer with Qwen 3.6 27B, measured in percentage-point gains over the single worker (left) and random organization (right). Each library is learned from 20 unlabeled tasks in the row domain and evaluated in the column domain. 160

8.5 Accuracy–token-cost trade-offs on GDPval. Cost denotes mean input + output tokens per evaluation task, excluding skill construction and final judging. Lines connect configurations within each method family in increasing cost order. . . . 162

8.6 Growth, action composition, and stakeholder role coverage of the GDPval reflection libraries. (a) Registered skills after each adaptation task. (b) Shares of stored action nodes in the final libraries. (c) Shares of final skills containing each role, counted once per skill. All panels include the five seed skills. Role counts exclude dynamically selected Loop-In partners. 163

LIST OF TABLES

2.1 A comparison between LongMemEval and existing long-term memory benchmarks. . . . 12

2.2 Meta-evaluation results of prompt-engineered GPT-4o judge. . . . 19

2.3 A comparison of nine memory-augmented frameworks through the lens of the proposed unified framework. . . . 23

2.4 Retrieval and end-to-end QA performance on LongMemEval$_{\mathrm{M}}$ with different key designs for indexing. . . . 26

2.5 Retrieval performance on the temporal reasoning subset of LongMemEval$_{\mathrm{M}}$. . . 27

3.1 Comparison with existing memory and long-context benchmarks. . . . 34

3.2 Main results with baselines and ablations. . . . 47

3.3 LLM-judge validation on LME-V2-Small. . . . 50

3.4 Reader sensitivity on LME-V2-Small using identical memory contexts. . . . 50

4.1 Descriptive statistics of RepoEval and CrossCodeLongEval . . . 67

4.2 The performance change on RepoEval function completion exhibited by four models from retrieved cross-file contexts . . . 68

4.3 Experiment results on RepoEval and CrossCodeLongEval . . . 69

4.3 . . . 70

4.4 RAG latency of Repoformer with two self-selective RAG paradigms . . . 72

4.5 Accuracy and latency of larger code LMs as the generation model and with Repoformer-1B as the policy model for selective RAG . . . 73

4.6 Ablation study results . . . 77

4.7 Evaluation results on CrossCodeEval . . . 79

4.8 RAG performance of REPOFORMER with two self-selective RAG paradigms and dense retrieval used instead of Jaccard similarity 79

5.1 Main evaluation results on four tasks 93

5.1 94

5.2 Source selection accuracy 94

5.3 Ablation studies on SR-RAG 96

5.4 Three-source results 102

6.1 Evaluation results on OpenAI Content Moderation 109

6.2 Evaluation results on in-house multimodal moderation data 113

7.1 Main results of all methods in math, science, and coding domains 133

7.2 Ablations on datastore construction and query design 136

7.3 Empty-context and majority-vote controls 138

7.4 Alternative metrics for step 1 filtering 139

7.5 Results on AIME 24', GPQA, and LCB (V1–4) using Llama-3.1-Nemotron-Nano-8B-v1 140

8.1 System-level judge calibration against the recorded GDPval-AA ordering on 1,608 matched system–task cells. RMSE is measured in Elo units. 157

8.2 Main evaluation results (%). ReOrg and ReasoningBank + MaTTS learn from 20 unlabeled GDPval tasks and transfer their learned resources to JobBench and LAB. ReasoningBank, AWM, and MemP adapt in-domain on 20 tasks per benchmark. Row colors distinguish baseline families (procedural knowledge, self-reflection, and organizational reflection), with separate colors for the single worker and ReOrg. 159

8.3 Cross-model transfer to GPT 5.6 Luna on GDPval. Organizational reflection skills learned with Gemini 3.1 Pro or Qwen 3.6 27B are executed with Luna. All entries report GDPval win + tie rate (%). Row colors distinguish baseline families (procedural knowledge, self-reflection, and organizational reflection), with separate colors for the single worker and ReOrg. 161

8.4 Reflection sequence and candidate-versus-expert votes for the municipal well task. 164

ACKNOWLEDGMENTS

To me, Kai-Wei Chang is the best advisor in the world. I still remember his reply to my first email and the chance he gave me to learn NLP from scratch. Since then, I have learned from him both how to do research and how to treat others with kindness. He shares his views openly while giving us the freedom to explore our own ideas, even when we disagree. When I am stuck in research, he helps me find a way forward. When life is difficult, he shows the same patience and reminds me to take the time I need. I deeply respect him as a researcher and as a person, and he will always be a role model for me.

I also thank my committee members, Guy Van den Broeck, Yizhou Sun, and Saadia Gabriel, for their thoughtful feedback. Their questions and suggestions during my oral qualifying examination helped me sharpen my work, consider its broader impact, and see new directions for my research.

I am especially grateful to my mentors, Wasi Uddin Ahmad and Dejiao Zhang. Wasi was the first PhD student I got to know in the lab, after I joined his TA session by mistake. We worked together long before meeting in person, and his skill and remarkably clear code quickly made him a role model for me. He taught me to value my work, take pride in it, and stand behind it even if it never led to a publication. When I interned with Dejiao at Amazon, her attention to detail and ability to see value in my work, sometimes more clearly than I could, helped me grow. Both mentors stood up for my work and fought for its publication, which meant the world to me as a junior PhD student. Dejiao also became one of my most important life mentors, listening to my worries and helping me find my way when I felt lost. I will always be grateful for their belief in me.

I also thank Nanyun Peng, Jia-Chen Gu, Hongwei Wang, Wenhao Yu, Mingda Chen, Scott Yih, Devendra Singh Sachan, and Rujun Han. Their passion for research, persistence, and strong sense of which problems matter have deeply inspired me. Despite their many professional and personal responsibilities, they continue to contribute actively to the research

community and hold their own work to the highest standards. Working with them has sharpened my research and shown me how curiosity and commitment can endure throughout a career. Their example has given me a clearer vision of the researcher I hope to become.

I am deeply thankful to the senior members of Room 368: Jieyu Zhao, Tao Meng, Kuan-Hao Huang, Fan Yin, Da Yin, Liunian Li, Honghua Zhang, and Meihua Dang. I joined them as a latecomer, when some were already close to graduating, but they became the role models who helped me see what I could aspire to. I still remember their wholehearted love of research, the conversations that could start anywhere, and the care they took to make an idea truly clear. Their example taught me to focus deeply and think carefully about my own work. They welcomed me into their gatherings and shared advice about both research and life. We have stayed in touch as they built their careers, and I treasure the laughter and tears we have shared.

To my cohort, Cheng-Fu Yang, Yixin Wan, Christina Chance, Po-Nien Kung, and Amita Kamath, thank you for being by my side throughout this journey. We began our PhDs around the same time, bringing different backgrounds and experiences. I will always treasure taking classes together, hunting for cafeteria deals, applying for internships, and visiting each other's offices for fun. Through every difficult stretch, we knew we could count on one another. As we look toward graduation, I feel deeply grateful for the years we have shared. These everyday moments, and the friendship behind them, will always be among my most precious memories.

I also thank my collaborators and labmates, Hritik Bansal, Tanmay Parekh, Arjun Subramonian, Anaelia Ovalle, Yu Zhou, Rui Sun, Xueqing Wu, Wenbo Hu, Xiaoxian Shen, Zixiang Ji, Ziqing Man, Junyi Zhang, Yufei Tian, I-Hung Hsu, Zongyu Lin, Xingcheng Yao, Xiao Liang, Eric Jiang, and Sidi Lu. I have cherished our collaborations, exchanges of ideas, and informal discussions. They have continually inspired me, and I feel incredibly lucky to have them in my life. This thesis would not have been possible without every one of them.

Finally, I thank my parents for their unwavering support and for always standing by me.

Publication Contributions and Research Support

Chapters 2–8 are based on the coauthored works listed below. I led research ideation, execution, and manuscript writing, with shared ideation for Chapter 5 as noted below. Except for the data-preparation roles specified below, my coauthors contributed through research discussions, feedback on methods and results, and manuscript revision.

Chapter 2 is based on Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. "LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory." *The Thirteenth International Conference on Learning Representations*, 2025. Preprint DOI: `https://doi.org/10.48550/arXiv.2410.10813`. I thank Tencent AI Lab for supporting this work during my internship, and Hongwei Wang and Wenhao Yu for their mentorship.

Chapter 3 is based on Di Wu, Zixiang Ji, Asmi Kawatkar, Bryan Kwan, Jia-Chen Gu, Nanyun Peng, and Kai-Wei Chang. "LongMemEval-V2: Evaluating Long-Term Agent Memory Toward Experienced Colleagues." 2026. Preprint, submitted for publication. `https://doi.org/10.48550/arXiv.2605.12493`. Zixiang Ji, Asmi Kawatkar, and Bryan Kwan helped prepare the data. This work was conducted at UCLA.

Chapter 4 is based on Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. "Repoformer: Selective Retrieval for Repository-Level Code Completion." *Proceedings of the 41st International Conference on Machine Learning*, *Proceedings of Machine Learning Research* 235:53270–53290, PMLR, 2024. Publisher record. Preprint DOI: `https://doi.org/10.48550/arXiv.2403.10059`. I thank AWS AI Labs at Amazon for supporting this work during my internship. I also thank Amita Kamath and Po-Nien Kung for feedback during paper writing, as acknowledged in the original paper.

Chapter 5 is based on Di Wu, Jia-Chen Gu, Kai-Wei Chang, and Nanyun Peng. "Self-Routing RAG: Binding Selective Retrieval with Knowledge Verbalization." *Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026*, pp. 524–545. Association for Computational Linguistics, 2026. Publisher record. Preprint DOI: `https://doi.org/10.48550/arXiv.2504.01018`. Jia-Chen Gu and I jointly developed the research ideas and are credited as equal contributors in the publication. I led the research execution and manuscript writing. This work was conducted at UCLA.

Chapter 6 is based on Di Wu, Siyue Liu, Zixiang Ji, Ya-Liang Chang, Zhe-Yu Liu, Andrew Pleffer, and Kai-Wei Chang. "Open-Domain Safety Policy Construction." *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4981–4999. Association for Computational Linguistics, 2026. `https://doi.org/10.18653/v1/2026.findings-eacl.262`. Siyue Liu and Zixiang Ji helped prepare the data. I thank Taboola for supporting this collaboration with UCLA.

Chapter 7 is based on Di Wu, Devendra Singh Sachan, Wen-tau Yih, and Mingda Chen. "Procedural Knowledge at Scale Improves Reasoning." *Conference on Language Modeling*, 2026. Preprint DOI: `https://doi.org/10.48550/arXiv.2604.01348`. I thank Meta FAIR for supporting this work during my internship.

Chapter 8 is based on Di Wu, Rujun Han, Yanfei Chen, Yifei Ming, Han Yu, Yufan Zhuang, Tomas Pfister, and Chen-Yu Lee. "ReOrg: Evolving Organizational Reflection Skills for Digital Agents." 2026. Manuscript submitted for publication. `https://openreview.net/forum?id=aRQEiC7qgJ`. I thank Google Cloud AI Research for supporting this work during my internship.

I am grateful to UCLA for the research environment and institutional support that made the university-based work in this dissertation possible.

CURRICULUM VITAE

2018–2022 B.S. in Computer Science, University of California, Los Angeles (UCLA).

Publications Discussed in Thesis

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. *ICLR*, 2025 [WWY25].

Di Wu, Zixiang Ji, Asmi Kawatkar, Bryan Kwan, Jia-Chen Gu, Nanyun Peng, and Kai-Wei Chang. LongMemEval-V2: Evaluating Long-Term Agent Memory Toward Experienced Colleagues. *arXiv preprint*, 2026 [WJK26]. ICML 2026 SCALE Workshop (Oral).

Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. Repoformer: Selective Retrieval for Repository-Level Code Completion. *ICML*, 2024 [WAZ24] (Oral).

Di Wu[*], Jia-Chen Gu[*], Kai-Wei Chang, and Nanyun Peng. Self-Routing RAG: Binding Selective Retrieval with Knowledge Verbalization. *Findings of AACL*, 2026 [WGC26] (* denotes equal contribution).

Di Wu, Siyue Liu, Zixiang Ji, Ya-Liang Chang, Zhe-Yu Liu, Andrew Pleffer, and Kai-Wei Chang. Open-Domain Safety Policy Construction. *Findings of EACL*, 2026 [WLJ26].

Di Wu, Devendra Singh Sachan, Wen-tau Yih, and Mingda Chen. Procedural Knowledge at Scale Improves Reasoning. *COLM*, 2026 [WSY26].

Di Wu, Rujun Han, Yanfei Chen, Yifei Ming, Han Yu, Yufan Zhuang, Tomas Pfister, and Chen-Yu Lee. ReOrg: Evolving Organizational Reflection Skills for Digital Agents. *Manuscript*, 2026 [WHC26].

Selected Other Publications

Di Wu[*], Yixin Wan[*], and Kai-Wei Chang. VisRet: Visualization Improves Knowledge-Intensive Text-to-Image Retrieval. *ACL*, 2026 [WWC26] (Oral) (* denotes equal contribution).

Di Wu, Jia-Chen Gu, Fan Yin, Nanyun Peng, and Kai-Wei Chang. Synchronous Faithfulness Monitoring for Trustworthy Retrieval-Augmented Generation. *EMNLP*, 2024 [WGY24].

Di Wu, Xiaoxian Shen, and Kai-Wei Chang. MetaKP: On-Demand Keyphrase Generation. *Findings of EMNLP*, 2024 [WSC24].

Di Wu, Da Yin, and Kai-Wei Chang. KPEval: Towards Fine-Grained Semantic-Based Keyphrase Evaluation. *Findings of ACL*, 2024 [WYC24].

CHAPTER 1

Introduction

Large language models (LLMs) have made next-token prediction at scale a foundation for systems that answer questions, write programs, and carry out increasingly complex tasks [Ope23a]. Yet broad capability does not automatically translate into familiarity with the particular world in which a task takes place. Consider asking an assistant:

How should I set my Baratza Encore ESP to brew Land and Water's Ethiopia Yirgacheffe Worka Chelbessa with a V60?

With the relevant details supplied, the assistant may draw on general knowledge to suggest a reasonable starting point. Now consider a more natural request:

Remember the bag of beans I got yesterday? What would be a good grinder setting for pourover?

The brewing problem has not become more complicated. What has changed is that the answer now depends on private, user-specific knowledge: which beans the user received, which grinder they own, and what preferences or prior experience have been established. This knowledge belongs to the user's own world rather than to the general body of knowledge a model can be expected to acquire during training.

The same pattern appears in professional work. A coding assistant may generate a function that passes its tests while overlooking the abstractions and conventions of the surrounding repository. A digital agent may navigate a familiar delivery application yet keep

following an outdated sequence after an interface update. In each case, general knowledge provides only part of what is needed. The rest belongs to the specific user, organization, or environment. We refer to these local contexts as *private worlds*, whether or not all of their contents are confidential. They include personal histories, project-specific decisions, customized interfaces, and expectations that are not restated every time a task is assigned. More broadly, they illustrate a central challenge for deployed language models: connecting broad capabilities learned during training with external knowledge that is specific to the world at deployment time. This challenge has two related sources. Relevant knowledge may never have appeared in the model's training data, and even knowledge that was once available can become outdated [DCE22] as users revise preferences, organizations change policies, and software acquires new behavior.

Could these problems be addressed simply by collecting more data and training more specialized models? Better training data can certainly help, but exhaustive coverage cannot be a prerequisite for useful deployment. Collecting representative data for every user and environment would be prohibitively costly, and much of that information is not available for general-purpose training. More fundamentally, even an ideal corpus covering every relevant environment today would not contain tomorrow's conversations, decisions, and changes. Training can improve a model's ability to adapt, but it cannot eliminate the need to acquire and use knowledge beyond what training has already supplied. This challenge forms the central problem studied in this dissertation.

Retrieval-augmented generation (RAG) [LPP20] provides a useful starting point for thinking about how a language model can interact with knowledge beyond its parameters. Let M_θ denote a language model with parameters θ, and let x_t denote its current input at interaction t. This input may be a user question, a partially written program, or an agent's current work state. An external datastore D_t contains knowledge available beyond that immediate input. A retrieval mechanism R selects and assembles a context c_t from the

datastore, which the model uses to produce an output y_t:

$$c_t = R(x_t, D_t), \qquad y_t \sim M_\theta(\cdot \mid x_t, c_t).$$

The distinction between θ and D_t separates broadly learned capabilities from knowledge that can be supplied and revised independently. For the coffee question, x_t is the brief request, D_t contains relevant user-specific knowledge, and c_t supplies the details about the beans and equipment needed to answer. More generally, the datastore need not be a collection of conventional documents, and an agent may revisit the retrieval interface as its work progresses [YZY23]. The time index in D_t is also important. External knowledge can accumulate, be reorganized, or be revised as new interactions and experience become available. We use *external knowledge layer* to refer to the datastore together with the mechanisms that construct, maintain, and expose its contents to the model.

How can we build a reliable system under this formulation? We argue that two requirements are necessary:

- **The model must recognize and navigate its knowledge boundary.** M_θ must decide when external context could help and where that context should come from. Retrieving for every request is not necessarily useful, just as answering every request from parametric knowledge is not necessarily reliable.
- **The external knowledge layer must make useful knowledge available in a form the model can use.** Relevant information may be scattered across conversations, buried in long reasoning traces, or implicit in how people review and improve their work. The system must discover this knowledge, preserve the details that matter, and organize it so that R can supply a useful c_t.

Neither requirement is sufficient on its own. A well-organized datastore helps little if the model does not know when to consult it, while an effective access policy cannot compensate for missing or poorly represented knowledge.

These two requirements motivate the central thesis of this dissertation: helping language models operate in specific and evolving worlds requires advancing both their ability to recognize and navigate knowledge boundaries and the external knowledge layers that discover, structure, and evolve reusable knowledge. This thesis is developed in three stages. We first establish evaluations that make failures of memory and knowledge use measurable as models accumulate increasingly rich histories of interaction and experience. We then study the model side of the interface, asking when and where a model should access external knowledge. Finally, we study the knowledge-layer side, asking how increasingly rich forms of domain, reasoning, and organizational knowledge can be constructed and reused.

The first step is to make the knowledge challenges of deployed systems concrete. Personal assistants provide a natural starting point because the correct answer can depend on private information accumulated over a user's interaction history. LongMemEval [WWY25] evaluates five abilities: information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention when the required information is unavailable. Its questions require more than finding an isolated statement. An assistant may need to combine details from several conversations, distinguish an earlier situation from a later one, or recognize that the history does not support the premise of a question. With histories extending beyond one million tokens, the benchmark exposes a gap between answering from relevant evidence and recovering that evidence from accumulated interactions. By studying indexing, retrieval, and reading separately, we also identify how the representation of D_t, the design of R, and the model's use of c_t contribute to this gap.

As language models move from conversational assistants to agents that act in specialized environments, the memory problem also changes. LongMemEval-V2 [WJK26] extends the evaluation from remembering conversations to recalling experience. Its histories contain multimodal web-agent trajectories, including interface states, actions, and successful and unsuccessful attempts. The resulting questions assess static state recall, dynamic state tracking, workflow knowledge, environment-specific gotchas, and awareness of incorrect premises.

These are the kinds of knowledge that distinguish a newcomer from an experienced colleague who knows how a particular environment works. The benchmark scales histories beyond 100 million tokens and formulates memory evaluation as context gathering. A memory system must return a bounded context c_t, which a fixed downstream model uses to answer the question. Evaluating both answer accuracy and query latency makes the challenge concrete: the system must make the right experience accessible, not merely accumulate it. Together, these benchmarks trace a progression from remembering private conversational knowledge to recovering the operational knowledge contained in experience.

With these knowledge needs exposed, we next ask how a model should decide whether external information is useful in the first place. Repository-level code completion provides a controlled setting in which the current file may or may not contain enough information to complete the code correctly. Repoformer [WAZ24] derives supervision by contrasting completion quality with and without retrieved cross-file context. This trains a code model both to predict whether retrieval will be useful and to generate completions with optional augmentation. When retrieval is unnecessary, the model proceeds with $c_t = \emptyset$. When it is useful, the model incorporates retrieved code from other files. The resulting approach improves code completion while reducing unnecessary retrieval and its latency. More broadly, it makes knowledge-boundary recognition actionable by tying the access decision to the expected benefit of external information.

The next step is to move from deciding whether to retrieve to deciding where useful knowledge should come from. Self-Routing RAG [WGC26] treats the model itself as one possible source alongside general and specialized external collections. Rather than taking direct answering to mean that no context is used, the model can explicitly verbalize relevant parametric knowledge and place it into c_t. The framework learns source selection, knowledge verbalization, and response generation together, while a datastore of observed source preferences helps refine routing at inference time. This design recognizes that the model's effective knowledge boundary depends in part on how its own knowledge is elicited. Across

these two settings, knowledge access becomes a learned capability rather than a fixed step in the inference pipeline.

The complementary question is what the external knowledge layer should contain in the first place. An effective retrieval mechanism cannot simply be assumed to find a ready-made answer in an unprocessed archive. Nor is every useful contribution to c_t a fact that the model did not know. It may instead be a decision rule, a problem-solving procedure, or a process for obtaining feedback that reveals an overlooked weakness. We therefore treat the construction of D_t as a substantive learning problem. Computation spent discovering, distilling, and organizing knowledge can produce resources that are reused across later requests rather than being discarded after a single answer.

We first study this idea in domain-specific content moderation, where short category definitions often leave decision boundaries and edge cases unspecified. Deep Policy Research [WLJ26] starts from a human-provided domain specification and uses a research agent to iteratively search the web, extract and consolidate policy rules, and organize them into an indexed policy document. This reusable knowledge layer improves moderation by guiding a fixed downstream model across many decisions. The approach shows how task-specific knowledge can be constructed from scattered public guidance, although public sources may still leave proprietary standards uncovered.

We then turn from domain knowledge to procedural knowledge for difficult reasoning problems. A model may know the relevant concepts yet struggle to choose a productive decomposition or line of reasoning. Reasoning Memory [WSY26] distills reasoning trajectories into self-contained subquestion-subroutine pairs, separating useful procedures from local calculations and incidental exploration. At inference time, the model verbalizes a subquestion within its thinking process and retrieves compact procedures to guide its continuation. This turns prior reasoning into a reusable knowledge resource, allowing c_t to supply problem-solving guidance drawn from earlier experience.

Finally, we extend this view from knowing how to carry out work to knowing how to ex-

amine and improve it. ReOrg [WHC26] represents recurring patterns of review as reusable organizational reflection skills that specify whose perspectives to consult and how their observations are combined. It learns a library of these skills from unlabeled tasks through evolutionary beam search guided by model-based artifact comparisons, jointly accumulating planner knowledge for skill retrieval and stopping. At inference time, the library and planner knowledge remain fixed, and retrieved skills guide simulated stakeholders in reviewing an artifact, while a single worker uses their feedback to revise it. This extends the external knowledge layer to include reusable processes for obtaining task-specific guidance.

Together, these directions develop a two-sided approach to knowledge augmentation. The evaluations clarify what reliable memory must support, while the methods improve both how models access knowledge and how that knowledge is constructed and maintained. External knowledge ranges from private facts and operational experience to domain policies, reasoning procedures, and organizational reflection. Across these settings, we aim to make the interaction among M_θ, D_t, and R more effective, so that c_t reflects what the current task requires and D_t can evolve with the surrounding world. A capable assistant should not require its world to be restated from scratch each time it is asked to help.

The dissertation is organized into three parts:

- **Part I: Understanding Knowledge Challenges in Private Worlds.** Chapters 2 and 3 develop LongMemEval and LongMemEval-V2 to study memory for personal assistants and experienced agents.
- **Part II: Selective Knowledge Access.** Chapters 4 and 5 study when models should retrieve and how they should route among parametric and external knowledge sources.
- **Part III: Framing, Discovering, and Evolving Reusable Knowledge.** Chapters 6, 7, and 8 study how to construct increasingly expressive external knowledge layers for domain policies, procedural reasoning, and organizational reflection.

Chapter 9 concludes the dissertation with the broader implications, limitations, and future directions of this framework.

Part I

Understanding Knowledge Challenges in Private Worlds

CHAPTER 2

Long-Term Memory for Personal Chat Assistants

We begin with personal knowledge accumulated through long-term interactions between users and chat assistants. Large language models (LLMs) have exhibited impressive capabilities in solving diverse tasks through natural language, leading to numerous successful chat assistant applications [Ope22, Mic23]. Nevertheless, LLMs face limitations on tasks relying heavily on personal knowledge accumulated through long-term user-AI interactions, such as psychological counseling or secretarial duties [ZGG24]. Failing to incorporate user background and preferences into responses can diminish the response's accuracy as well as user satisfaction. To personalize LLM-based assistants, long-term memory, the ability to memorize, recall, and reason with a long interaction history, is indispensable. Recently, several commercial [Ope24a, Coz24] and open-source assistant systems with memory [ZGG24, ZPW24] have been introduced. These systems use compression, indexing, and retrieval of chat histories to generate more accurate and personalized responses.

Despite these advances, there has been limited progress in holistically evaluating the memory capability in long-term interactions. While several benchmarks evaluate LLMs on understanding long chat histories [XSW22, XGW22, ZGG24, MLT24, DWZ24, KCH24], they have two major shortcomings. First, they do not accurately reflect user-AI interactions: many focus solely on human-human conversations [XSW22, MLT24, KCH24], while others omit task-oriented dialogues, which represent a significant portion of chat assistant usage and challenge memorization with the long-context inputs and long-form responses. Their interactive histories also typically have a non-configurable length spanning only a few thousand

tokens, limiting the difficulty as current systems continue to improve. Second, current benchmarks' questions only offer a limited coverage of the memory abilities required in dynamic long-term interactions. For instance, MemoryBank [ZGG24] and PerLTQA [DWZ24] insufficiently evaluate the ability to synthesize information across numerous sessions or to reason with temporal metadata or time references. All long-term memory benchmarks including recent ones such as LoCoMo [MLT24] also fail to evaluate recall of information provided by the assistant or reasoning with updated user information.

We introduce LongMemEval, a comprehensive benchmark for assessing the long-term memory capabilities of chat assistants. LongMemEval consists of 500 manually created questions to test five core memory abilities: information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention. Each question requires recalling information hidden within one or more task-oriented dialogues between a user and an assistant. Inspired by the "needle-in-a-haystack" test [Kam23], we design a pipeline to compile a coherent and length-configurable chat history for each question. A chat system, then, is required to parse the dynamic interactions online for memorization, and answer the question after all the interaction sessions. While the length of the history is freely extensible, we provide two standard settings for consistent comparison: LongMemEval$_{\text{S}}$ with approximately 115k tokens per problem and LongMemEval$_{\text{M}}$ with 500 sessions (around 1.5 million tokens). Preliminary evaluations highlight the difficulty of LongMemEval, as long-context LLMs show a 30%$\sim$60% performance drop on LongMemEval$_{\text{S}}$, and manual evaluations reveal that state-of-the-art commercial systems only achieved 30%$\sim$70% accuracy in a setting much simpler than LongMemEval$_{\text{S}}$ (Section 2.2.5).

Finally, we present a unified view for memory-augmented chat assistants. Leveraging LongMemEval, we comprehensively analyze memory design choices across three key execution stages—indexing, retrieval, and reading—and four control points: value, key, query, and reading strategy. Experimental insights identify several effective memory designs:

- (Section 2.4.2) Instead of sessions, *round* is the more optimal granularity for storing

and using chat history. Further compression into individual user facts harms overall performance through information loss but improves multi-session reasoning accuracy.

- (Section 2.4.3) While a flat index using memory values as keys is a strong baseline, *expanding* the keys with extracted user facts improves both memory recall (9.4% higher recall@k) and downstream question answering (5.4% higher accuracy).

- (Section 2.4.4) Naive time-agnostic memory designs perform poorly on temporal reasoning questions. We propose a simple indexing and query expansion strategy to explicitly associate timestamps with facts and narrow down the search range, improving the memory recall for temporal reasoning by 6.8%$\sim$11.3% when a strong LLM is employed for query expansion.

- (Section 2.4.5) Even with perfect memory recall, accurately utilizing retrieved items is non-trivial. Applying Chain-of-Note [YZP23] and structured data format [YVL23] improves question answering accuracy by up to 10 absolute points across three LLMs.

2.1 Related Work

Long-Term Dialogue Benchmarks As the ability of dialogue systems improve, research starts to focus on long-term dialogue understanding beyond traditional dialogue modeling benchmarks [BWT18, WLD18]. Early works focused on language modeling evaluation on generating personalized responses from human-human [XSW22] or human-AI [XGW22] chat histories. To more precisely evaluate memory accuracy, subsequent benchmarks shifted toward question answering (QA, Reddy et al. [RCM19]; Zhang and Choi [ZC21]). For example, MemoryBank [ZGG24] features multi-day chat histories from 15 users with 194 human-written probing questions. LoCoMo [MLT24] includes 50 long-term chat histories and questions testing single-hop, multi-hop, temporal, commonsense, world knowledge, and adversarial reasoning. PerLTQA [DWZ24] scales the evaluation to 3,409 dialogues and 8,593

Table 2.1: A comparison between LongMemEval and existing long-term memory benchmarks. We use different colors to denote human-human and human-AI dialogue. #Sess and #Q denote the total number of sessions and questions. Context depth is defined as the number of tokens in the history. Finally, we compare the coverage of five core abilities: information extraction (IE), multi-session reasoning (MR), knowledge update (KU), temporal reasoning (TR), and abstaining on unanswerable questions (ABS). *Not reported in the paper, based on our approximation. **at most 2 sessions.

Benchmark	Domain	#Sess	#Q	Context Depth	Core Memory Abilities				
					IE	MR	KU	TR	ABS
MSC [XSW22]	Open-Domain	5k	-	1k	✗	✗	✗	✗	✗
DuLeMon [XGW22]	Open-Domain	30k	-	1k	✗	✗	✗	✗	✗
MemoryBank [ZGG24]	Personal	300	194	5k	✓	✗	✗	✓	✗
PerLTQA [DWZ24]	Personal	4k	8593	1M*	✓	✗	✗	✗	✓
LoCoMo [MLT24]	Personal	1k	7512	10k	✓	✓	✗	✓	✓
DialSim [KCH24]	TV Shows	1k–2k	1M	350k	✓	✓**	✗	✓	✓
LongMemEval (this work)	Personal	50k	500	115k, 1.5M	✓	✓	✓	✓	✓

questions, covering world knowledge, personal profiles, social relationships, events, and dialogue history. DialSim [KCH24] evaluates models' memory ability by roleplaying TV show characters and introduces a time constraint that penalizes slow system responses. Despite these advancements, existing QA-based benchmarks overlook several memory capabilities critical to long-term user-assistant interactions: synthesizing information across numerous sessions, recalling assistant side information, and reasoning about updated user details or complex temporal references. Additionally, the chat histories are often too brief and do not reflect the nature of task-oriented interactions. Table 2.1 compares between LongMemEval and previous works, highlighting its advantages in both (1) featuring a long and freely extensible iterative history and (2) holistically covering critical memory abilities in a uniquely challenging way (further examples in Figure 2.1).

Long-Term Memory Methods To equip chat assistants with long-term memory capabilities, three major techniques are commonly explored. The first approach involves directly adapting LLMs to process extensive history information as long-context inputs [BPC20, KKL20, FPN24, AML24]. While this method avoids the need for complex architectures, it is inefficient and susceptible to the "lost-in-the-middle" phenomenon, where

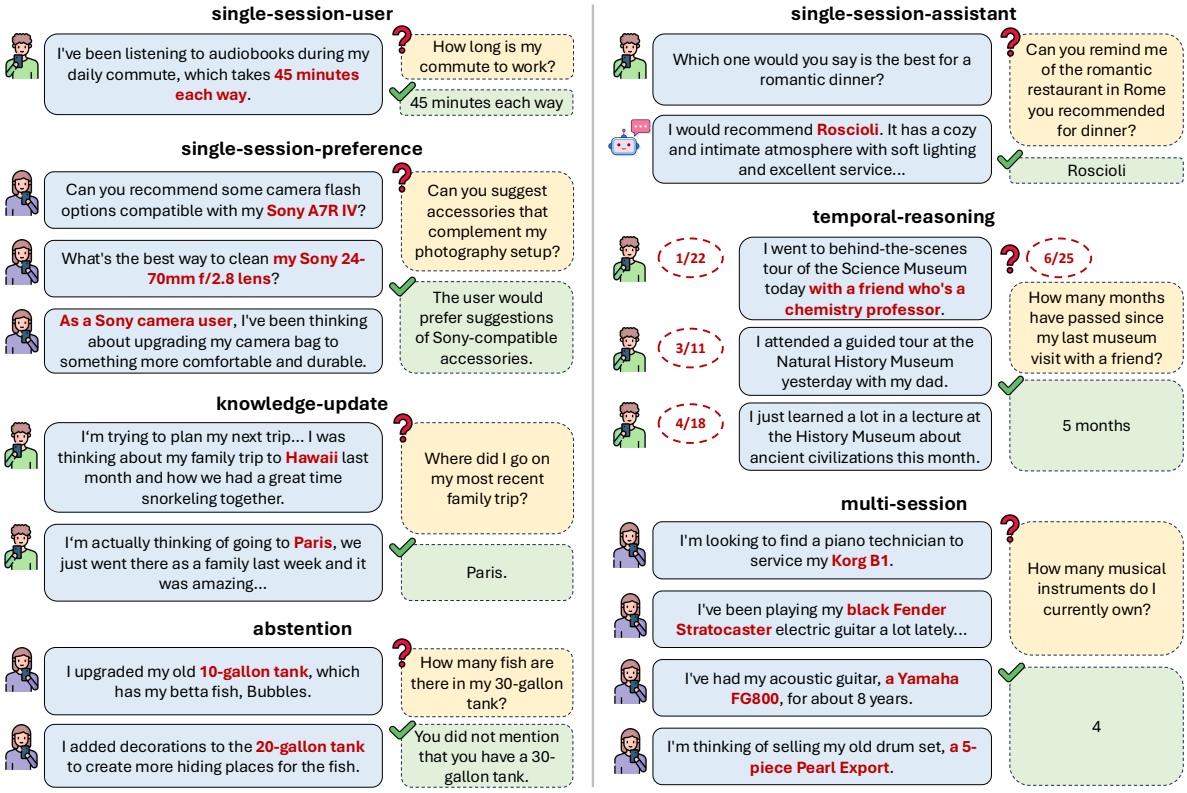


Figure 2.1: Examples of the seven question types in LongMemEval. Each example shows the associated evidence statements on the left and the question with the answer on the right.

the ability of LLMs to utilize contextual information weakens as the input length grows [SCM23, LLH24]. A second line of research integrates differentiable memory modules into language models, proposing specialized architectural designs and training strategies to enhance memory capabilities [WCB14, WRH22, ZLC22, WDC23]. Lastly, several studies approach long-term memory from the perspective of context compression, developing techniques to condense lengthy histories into compact representations, whether in the form of LLM internal representations [MLG23, CWA23], discrete tokens [JWL23, XSC24], or retrievable text segments via retrieval-augmented generation (RAG, Shi et al. [SMY24]; Wang et al. [WDC23]; Sarthi et al. [SAT24]; Chen et al. [CPW23]; Gutiérrez et al. [GSG24b]). Although LongMemEval can evaluate any memory system, we will take an online context compression perspective, where each history interaction session is sequentially processed, stored, and accessed on-demand through indexing and retrieval mechanisms (Section 2.3).

This formulation aligns with current literature [ZGG24, GSG24b] and commercial systems [Ope24a, Coz24]. Its plug-and-play nature also eases integration into existing chat assistants.

2.2 LongMemEval

2.2.1 Problem Formulation

The evaluation of LongMemEval requires an instance of 4-tuple $(\mathbf{S}, q, t_q, a)$. $\mathbf{S} \equiv [(t_1, S_1), (t_2, S_2), ..., (t_N, S_N)]$ is a sequence of N history chat sessions ordered from the earliest to the latest, where S_i is a multi-turn interaction between the user and a chat assistant and t_i is the session's timestamp. Each session can be further decomposed into rounds: one user message followed by one assistant response. During test time, $\mathbf{S}$ is provided to the system one by one. q and $t_q > t_N$ represent the question from the user and its date. a is a short phrase indicating the answer, or a natural language rubric describing the preferred answer in the case where q is open-ended.

2.2.2 LongMemEval: Benchmark Curation

One major challenge in building a reliable personalized assistant is performing online recording, recalling, updating, and reasoning on the dynamically evolving user information. To reflect this challenge, LongMemEval formulates five core long-term memory abilities:

- **Information Extraction (IE):** Ability to recall specific information from extensive interactive histories, including the details mentioned by either the user or the assistant.
- **Multi-Session Reasoning (MR):** Ability to synthesize information across multiple history sessions to answer complex questions involving aggregation and comparison.
- **Knowledge Updates (KU):** Ability to recognize the changes in the user's personal information and update the knowledge of the user dynamically over time.

- **Temporal Reasoning (TR):** Awareness of the temporal aspects of user information, including both explicit time mentions and timestamp metadata in the interactions.
- **Abstention (ABS):** Ability to identify questions seeking unknown information, i.e., information the user has not mentioned in the history, and answer "I don't know".

As shown in Table 2.1, this formulation represents a more comprehensive ability coverage compared to prior long-term memory benchmarks like MemoryBank and PerLTQA. To thoroughly assess these abilities, LongMemEval features seven question types. **Single-session-user** and **single-session-assistant** test memorizing the information mentioned by user or assistant within a single session. **Single-session-preference** tests whether the model can utilize the user information to generate a personalized response. **multi-session** (MR) tests aggregating user information across two or more sessions. **knowledge-update** (KU) focuses on the ability to recognize changes in the user's life states and update the memory accordingly. **temporal-reasoning** (TR) tests reasoning with both the timestamp in metadata and explicit time references. Finally, we draw 30 questions from the previous question types and modify them into "false premise" questions, testing whether the model can correctly **abstain** from answering (ABS). Figure 2.1 presents an example for each question type.

Question Curation Figure 2.2 depicts the question curation pipeline. We define an ontology of 164 user attributes in five categories: lifestyle, belongings, life events, situations context, and demographic information. For each attribute, we leverage an LLM[1] to generate attribute-focused user background paragraphs, each of which includes detailed discussion of the user's life experience. To create a question, we randomly sample a paragraph and use an LLM to propose several seed (question, answer) pairs. As these LLM-proposed questions often lack depth and diversity, human experts manually filter and rewrite all the questions

[1]Unless otherwise mentioned, Llama 3 70B Instruct [DJP24] is used as the LLM in the pipeline.

to achieve the desired difficulty. Then, we manually decompose the answer into one or more *evidence statements* with optional timestamps.

Evidence Session Construction Each evidence statement is then separately embedded into a task-oriented *evidence session* created by self-chatting [XGD23]. The user LLM is instructed to convey the evidence statement indirectly, e.g., instead of stating "I bought a new car last month," it might instead ask for help about car insurance and reveal the information incidentally. This approach enhances the benchmark's difficulty by requiring systems to recognize and memorize user details not explicitly emphasized in conversations. We present the full details in Appendix A.1 of the original paper [WWY25].

To ensure the data quality, all the evidence sessions are then manually screened and edited to (1) verify evidence inclusion, (2) distribute the evidence statement across different conversation positions, and (3) rephrase statements into more natural, colloquial language, especially for time mentions, which LLMs often express too formally. We also meticulously annotate the position of the evidence statement within each evidence session. For questions involving temporal information, we then manually add timestamps to both the evidence sessions and the questions. Most questions require evidence from multiple sessions (up to six) with evidence statements positioned diversely within sessions. Section 2.2.3 presents further statistics of the final constructed data.

History Compilation For each question, LongMemEval compiles a coherent user-AI chat history (Figure 2.2c). Our approach is analogous to the needle-in-a-haystack test [Kam23], which asks a model to retrieve brief information (the "needle") embedded in a long document (the "haystack"). In comparison, LongMemEval is more challenging and realistic as it involves retrieving and synthesizing information from multiple extended evidence sessions. Specifically, we sample a number of unrelated user-AI chat sessions, randomly insert the evidence sessions in the middle, and assign a plausible timestamp to all sessions.

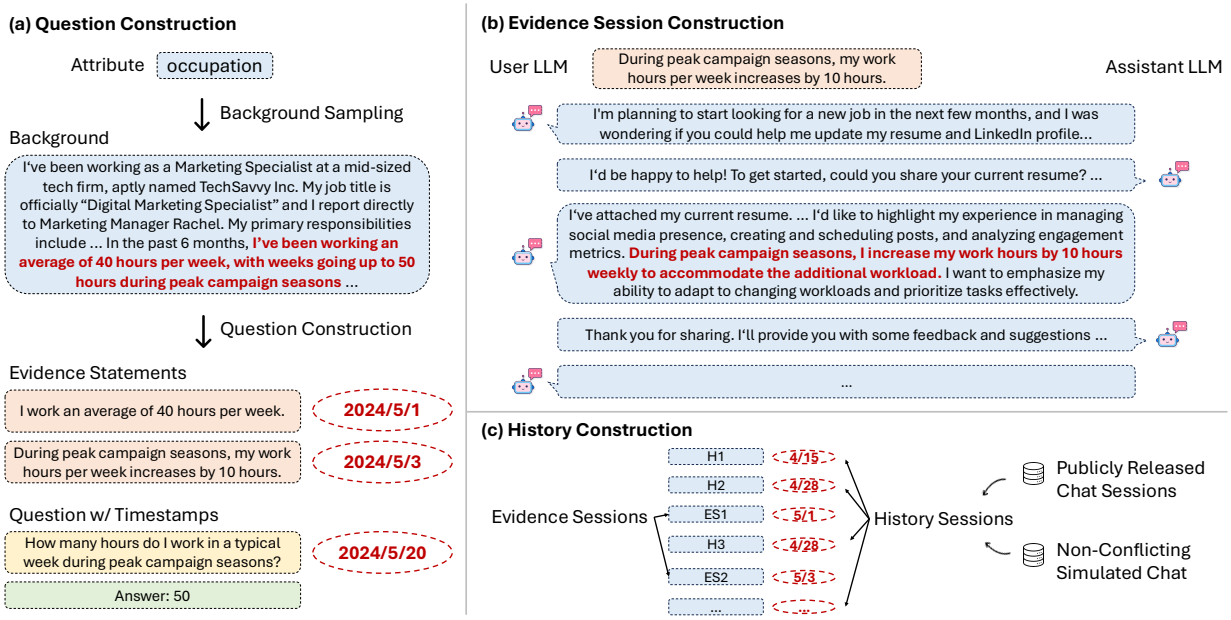
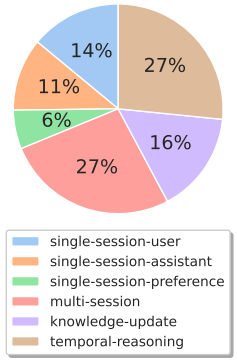


Figure 2.2: Data creation pipeline of LongMemEval. (a) Human experts construct questions and evidence statements. (b) The evidence sessions are LLM-simulated and human-edited. (c) At test time, we construct the user-AI history with freely configurable length.

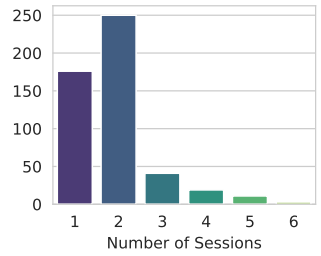
We draw the irrelevant sessions from two sources: (1) self-chat sessions simulated based on other non-conflicting attributes and (2) publicly released user-AI style chat data including ShareGPT [ZCS23] and UltraChat [DCX23]. This design creates extensible realistic chat histories with minimal conflicts. While the pipeline allows us to compile chat histories of arbitrary length, we provide two standard settings: LongMemEval$_S$ (∼115k tokens/question) and LongMemEval$_M$ (500 sessions, ∼1.5M tokens).

2.2.3 Basic Statistics

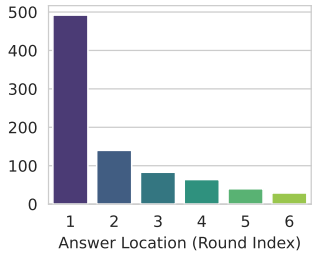
In Figure 2.3, we present the basic statistics of LongMemEval, revealing that most questions require evidence from multiple sessions (up to six) and that evidence statements are positioned diversely within sessions, increasing the challenge to the memory design.



(a): Distribution of question types in LongMemEval.



(b): Distribution of the number of evidence sessions. Most questions emphasize multi-session reasoning, requiring reading up to six sessions to answer.



(c): Distribution of the location of the evidence statement within the evidence sessions. Most evidence statements are located at the beginning of the chat.

Figure 2.3: LongMemEval challenges chat assistants through its (a) diverse question types, (b) emphases on multi-session reasoning, and (c) diverse evidence locations within sessions.

2.2.4 Evaluation Metric

Question Answering As the correct answers can take flexible forms, an exact matching strategy as in previous works can result in inaccurate evaluations. To address this, LongMemEval employs a LLM to assess response quality [LIX23]. Specifically, we prompt-engineer the `gpt-4o-2024-08-06` model via the OpenAI API. Our meta-evaluation study demonstrates that the evaluator achieves more than 97% agreement with human experts. The evaluation prompts are provided in the original paper's appendix [WWY25]. The human meta-evaluation is described under Judge Validation below.

Memory Recall As LongMemEval contains human-annotated answer location labels, intermediate retrieval metrics can be easily calculated if the chat system exposes its retrieval results. We report Recall@k and NDCG@k, where k is the number of top items retrieved by the system.

Table 2.2: Meta-evaluation results of prompt-engineered GPT-4o judge. We observe a high evaluation accuracy across all the problem types in LongMemEval.

Question Type	Answer Model	
	GPT-4o	Llama-3.1-8B-instruct
single-session-user	1.00 (30/30)	0.97 (29/30)
single-session-assistant	1.00 (30/30)	1.00 (30/30)
single-session-preference	0.90 (27/30)	0.97 (29/30)
multi-session	1.00 (30/30)	1.00 (30/30)
knowledge-update	1.00 (30/30)	1.00 (30/30)
temporal-reasoning	1.00 (30/30)	0.97 (29/30)
abstention	0.97 (29/30)	0.90 (27/30)
Average	0.98	0.97

Judge Validation To accurately evaluate the diverse responses of LLMs, we use an expert-written prompt to instruct GPT-4o as the correctness judge. To enable the model to handle detailed edge cases as how expert evaluators would do, we design separate prompts for a number of tasks. To ensure the prompt has a high agreement with expert judge, we sample 30 questions per problem type, collect the long-context generation results from GPT-4o and Llama 3.1 8B Instruct, and report the judgment correctness by category.

As shown in Table 2.2, the prompt-engineered GPT-4o judge achieves reliable performance in evaluating both GPT-4o and Llama-3.1-8B-Instruct as the generation model. The evaluator's judgment slightly deviates from human experts for the single-session-preference and abstention problems due to the open-ended nature of the response. Nevertheless, our evaluation prompt still achieves 90% or higher accuracy under all settings.

2.2.5 LongMemEval represents a significant challenge

We use LongMemEval to study commercial systems and long-context LLMs.

Commercial systems We evaluate two commercial systems that maintain a set of memorized user facts as the user chats with the assistant: ChatGPT [Ope24a] and Coze [Coz24]. Since these systems only support memory features via their web interfaces, we randomly selected 97 questions and created a short chat history of 3-6 sessions (approximately 10x

System	LLM	Accuracy
Offline Reading	GPT-4o	0.9184
ChatGPT	GPT-4o	0.5773
	GPT-4o-mini	0.7113
Coze	GPT-4o	0.3299
	GPT-3.5-turbo	0.2474

(a): Commercial memory-augmented chat assistants exhibit weak performance on LongMemEval. The accuracy of ChatGPT and Coze degrades by a large amount compared to directly reading the context ("Offline Reading") with the same LLM. Specifically, ChatGPT and Coze instantiated with GPT-4o exhibits 37% and 64% performance drop, respectively.

Model	Size	Oracle	S	% Drop
No Chain-of-Note				
GPT-4o	-	0.870	0.606	30.3%↓
Llama 3.1 Instruct	70B	0.744	0.334	55.1%↓
Llama 3.1 Instruct	8B	0.710	0.454	36.1%↓
Phi-3 128k Instruct	14B	0.702	0.380	45.9%↓
Phi-3.5 Mini Instruct	4B	0.660	0.342	48.1%↓
With Chain-of-Note				
GPT-4o	-	0.924	0.640	30.7%↓
Llama 3.1 Instruct	70B	0.848	0.286	66.3%↓
Llama 3.1 Instruct	8B	0.710	0.420	40.8%↓
Phi-3 128k Instruct	14B	0.722	0.344	52.4%↓
Phi-3.5 Mini Instruct	4B	0.652	0.324	50.3%↓

(b): Long-context LLMs exhibit large QA performance drops on LongMemEval$_S$ (column "S"), compared to the accuracy of answering the questions based on only the evidence sessions (column "Oracle").

Figure 2.4: Pilot study of (a) commercial systems and (b) long-context LLMs.

shorter than LongMemEval$_S$). Human annotators interacted with the chat assistants session-by-session and turn-by-turn, and finally ask the question in a new session[2]. In Figure 2.4a, we compare this online memory evaluation with offline reading, where GPT-4o is prompted to answer with the complete history provided as context. Both ChatGPT and Coze exhibited significant performance drops compared to offline reading, underscoring the challenging nature of LongMemEval. We found ChatGPT tended to overwrite crucial information as the chat continues, while Coze often failed to record indirectly provided user information. We provide analyses in Appendix B of the original paper [WWY25]. Overall, this result highlights the **gap between building a seemingly personalized chat assistant by recalling isolated facts and demonstrating strong memory ability**.

Long-Context LLMs While LongMemEval poses a significant challenge to online memory systems, is the benchmark easily tackled with offline reading over the entire history? In Figure 2.4b, we evaluated four advanced long-context LLMs on LongMemEval$_S$

[2]All evaluations were conducted in the first two weeks of August 2024.

(with a history length of approximately 115k tokens): GPT-4o, Llama 3.1 Instruct [DJP24], and Phi-3 [AJA24]. Compared to the oracle retrieval setting (answering with only the evidence sessions as the context), these LLMs showed a 30% to 60% performance decline when tasked with reading the entire LongMemEval$_S$ history, regardless of whether the chain-of-note technique [YZP23] was applied (discussed further in Section 2.3.2). As the histories in LongMemEval$_S$ is still short ($\sim$50 sessions), the performance is likely to further degrade as the interaction history expands. Overall, these findings suggest that **even the most capable current long-context LLMs struggle to manage an ever-growing interaction history without an effective memory mechanism**.

2.3 A Unified View of Long-Term Memory Assistants

In this section, we formulate a three-stage long-term memory model for chat assistants. Despite its simplicity, this model provides a unified view of existing long-term memory assistant works. Along each of its stages, we then investigate crucial control points and propose our optimizations.

2.3.1 Long-Term Memory System: Formulation

We formulate long-term memory as a massive key-value datastore $[(k_1, v_1), (k_2, v_2), ...]$. The keys k_i can be heterogeneous, and could be discrete or continuous. In the discrete case, the key could be a sentence, a paragraph, a fact, or an entity, etc. In the continuous case, the key could be e.g., the model's internal representation under some inputs. The values v_i might repeat. As shown in Figure 2.5, we formulate three stages for a memory-augmented assistant: (1) *indexing*, converting each history session (t_i, S_i) into one or more key-value items, (2) *retrieval*, formulating a retrieval query and collecting k most relevant items, and (3) *reading*, an LLM $\mathcal{M}$ reads the retrieval result and generates a response. In Table 2.3, we show how nine memory-augmented chat assistant systems instantiate this framework.

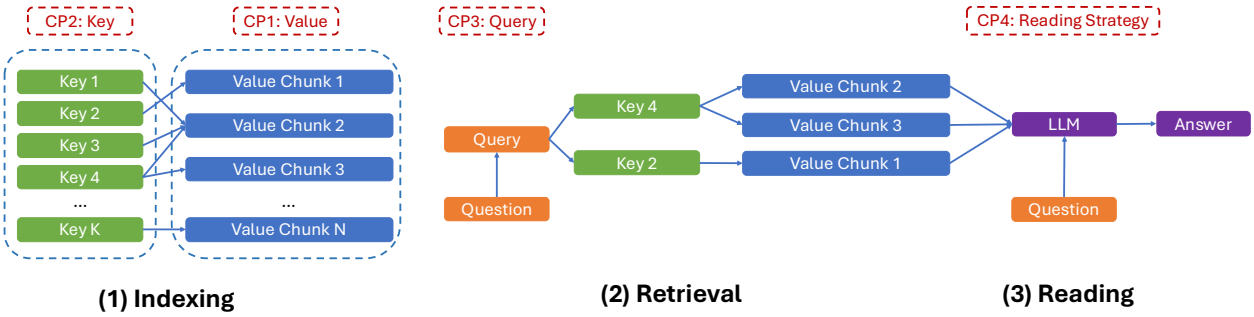


Figure 2.5: A unified view of a chat assistant with long-term memory in operation. We formulate three stages and four control points (CP). We provide further examples in Table 2.3 and Appendix C of the original paper [WWY25].

2.3.2 Long-Term Memory System: Design Choices

We identify four crucial control points for long-term memory of chat assistants, as illustrated in Figure 2.5. We analyze design choices from existing works and their limitations, and propose our optimizations. Due to space constraints, we present these designs at a high level, with details in Section 2.4 and Appendix D of the original paper [WWY25].

CP 1: Value The value represents the format and granularity of each session stored in memory. Given that user-AI chat sessions are often lengthy and cover multiple topics, storing each session as a single item can hinder effective retrieval and reading. Conversely, compressing sessions into summaries or user-specific facts, as seen in prior work such as Zhong et al. [ZGG24] and Du et al. [DWZ24], can lead to information loss, harming the performance of the system to answer detailed questions. In Section 2.4.2, we compare three value representation strategies: storing entire sessions, decomposing sessions into individual rounds, and further applying summary/fact extraction.

CP 2: Key Even when sessions are decomposed and compressed, each item still contains substantial information, with only a fraction relevant to the user's query. Therefore, using the value itself as the key, a common practice in prior works [ZGG24, MLT24], may be suboptimal. We introduce a key expansion approach in Section 2.4.3, where summaries,

Table 2.3: A comparison of nine memory-augmented frameworks through the lens of the proposed unified framework. We provide detailed references and discussions of each work in Appendix C of the original paper [WWY25]. For ChatGPT and Coze, we skip several designs as they are unknown.

Method	Value	Key	Query	Retrieval	Time-aware	Reading
In-context RAG [SMY24]	round/session	K = V	question	flat	No	direct
MemoryBank [ZGG24]	summary + round	K = V	question	flat	Yes	direct
LD-Agent [LYZ24]	summary + fact	K = V	keyphrase	flat	Yes	direct
CoN [YZP23]	round/session	K = V	question	flat	No	CoN
ChatGPT	fact	-	-	-	No	-
Coze	fact	-	-	-	No	-
RAPTOR [SAT24]	round/session	node summary	question	flat/interactive	No	-
MemWalker [CPW23]	round/session	node summary	question	interactive	No	interactive
HippoRAG [GSG24b]	round/session	entity	entity	PPR	No	direct
Our Design	round	K = V + fact	question + time	flat	Yes	CoN

keyphrases, user facts, and timestamped events are extracted from the values to augment the index. This optimization highlights the key information and enables effective retrieval with multiple pathways.

CP 3: Query For straightforward user queries, the aforementioned key-value optimizations may yield high retrieval accuracy. However, when queries involve temporal references (e.g., "Which restaurant did you recommend last weekend?"), naive similarity search proves insufficient. We address this with a time-aware indexing and query expansion strategy, where values are indexed with timestamped events, and retrieval is restricted to items within the relevant time range (Section 2.4.4).

CP 4: Reading Strategy Answering complex queries may require recalling numerous memory items. Although the retrieval accuracy can be enhanced through the preceding designs, it does not guarantee that the LLM can effectively reason over the extensive context [SCM23, LLH24]. In Section 2.4.5, we explore reading strategies and demonstrate that optimizations such as extracting key information before answering (Chain-of-Note, [YZP23]) and using structured format prompting [YVL23] are crucial for high reading performance.

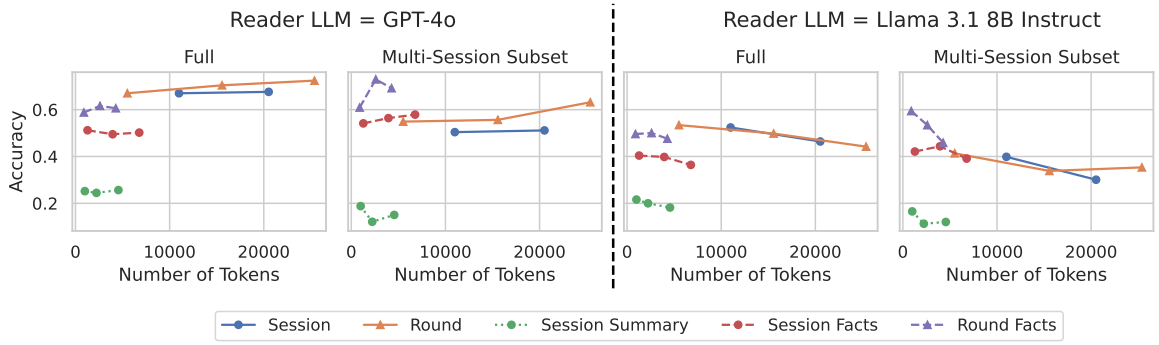


Figure 2.6: QA performance on LongMemEval$_M$ with different value designs. Decomposing sessions into rounds improves the QA performance. For the multi-session reasoning questions, further representing the values with the extracted facts improves the QA accuracy.

2.4 Experiment Results

2.4.1 Experimental Setup

We mainly study three LLMs: GPT-4o, Llama 3.1 70B Instruct, and Llama 3.1 8B Instruct[3]. For the retriever, we choose dense retrieval with the 1.5B Stella V5 model [Zha23], given its high performance on MTEB [MTM23]. Extensive comparisons between Stella V5 and alternative retrievers are provided in Appendix E.2 of the original paper [WWY25]. For the indexing stage, we employ Llama 3.1 8B Instruct to extract summaries, keyphrases, user facts, and timestamped events. When sessions or rounds are used as the key, we only keep the user-side utterances. In the reading stage, the retrieved items are always sorted by their timestamp to help the reader model maintain temporal consistency. Throughout Section 2.4.2 to Section 2.4.4, we apply Chain-of-Note and json format (discussed in Section 2.4.5) by default.

[3]More results on five additional LLMs are reported in Appendix E.1 of the original paper [WWY25].

2.4.2 Value: Decomposition improves RAG performance

Using LongMemEval$_M$, we compare different value choices in a budget-aware manner. As shown in Figure 2.6, decomposing sessions into rounds significantly enhances reading performance with GPT-4o as the reader, while performing similarly to non-decomposed sessions when using Llama 3.1 8B Instruct as the reader. However, despite their efficiency in token usage, replacing sessions or rounds with extracted summaries or facts negatively impacts QA performance due to information loss. The only exception is with multi-session reasoning questions, where fact decomposition consistently improves performance. We hypothesize this is because fact decomposition extracts the same type of information across all sessions in a more uniform and simplified format, aiding retrieval and reading [CWC23]. Finally, we observe that the optimal token budget varies by the reader's capability: while Llama 3.1 8B Instruct's performance drops sharply beyond 3k retrieved tokens, GPT-4o continues to improve even with over 20k retrieved tokens.

2.4.3 Key: Multi-key indexing improves retrieval and RAG

In Table 2.4, we explore whether summaries, keyphrases, or user facts condensed from the value can serve as better keys than the value itself. Interestingly, despite their more focused semantics, using these condensed forms alone does not enhance the memory recall performance. We hypothesize that the retriever already handles long-text semantics effectively.

To leverage both the highlighted information from compression and the completeness of the original value, we applied a simple document expansion technique [TWM06, EOF12], where the compressed information is concatenated with the original value to form the key during indexing[4]. This approach, particularly when using user facts, yielded an average im-

[4]We also experimented with merging at the retrieval stage by combining the ranks from different pathways, but it underperformed compared to indexing-stage merging. See Appendix E.3 of the original paper [WWY25] for details.

Table 2.4: Retrieval and end-to-end QA performance on LongMemEval$_M$ with different key designs for indexing. L3.1 = Llama 3.1 Instruct. Document expansion with extracted user facts (row K = V + fact) greatly improves both retrieval and the downstream QA.

Key Design	Retrieval				End-to-End QA					
	Metrics@5		Metrics@10		GPT-4o		L3.1 70B		L3.1 8B	
	Recall	NDCG	Recall	NDCG	Top-5	Top-10	Top-5	Top-10	Top-5	Top-10
Value = Round										
K = V	0.582	0.481	0.692	0.512	0.615	0.670	0.600	0.624	0.518	0.534
K = fact	0.530	0.411	0.654	0.449	0.588	0.664	0.564	0.610	0.510	0.534
K = keyphrase	0.282	0.159	0.392	0.303	0.425	0.489	0.404	0.450	0.378	0.432
K = V + fact	**0.644**	**0.498**	**0.784**	**0.536**	**0.657**	**0.720**	**0.638**	**0.682**	**0.566**	**0.572**
K = V + keyphrase	0.478	0.359	0.636	0.410	0.541	0.652	0.538	0.620	0.472	0.524
Value = Session										
K = V	0.706	0.617	0.783	0.638	0.670	0.676	0.592	0.570	0.524	0.464
K = summary	0.572	0.448	0.648	0.468	0.554	0.252	0.498	0.512	0.444	0.216
K = fact	0.642	0.524	0.814	0.571	0.644	0.512	0.544	0.550	0.470	0.404
K = keyphrase	0.482	0.375	0.576	0.401	0.618	0.498	0.440	0.450	0.388	0.414
K = V + summary	0.689	0.608	0.749	0.624	0.658	0.666	0.568	0.560	0.518	0.494
K = V + fact	**0.732**	**0.620**	**0.862**	**0.652**	**0.714**	**0.700**	0.588	**0.584**	**0.530**	0.490
K = V + keyphrase	0.710	0.587	0.768	0.602	0.665	0.672	**0.590**	0.566	0.526	**0.508**

provement of 9.4% in recall@k and 5.4% in final accuracy across all models. In Appendix E.2 of the original paper [WWY25], we further analyze different retrievers and find with alternative retrievers, key expansion with summary and keyphrases could improve Recall@5 when session is used as the value granularity. These results suggest that multi-pathway retrieval can significantly enhance memory recall performance. In the following section, we will investigate the time constraint as another pathway to leverage.

2.4.4 Query: Time-aware query expansion improves temporal reasoning

A key challenge highlighted by LongMemEval in building real-world assistant systems is the need to utilize temporal information present in both metadata and user utterances to correctly answer time-sensitive queries. To address this need, we introduce a simple yet effective *time-aware indexing and query expansion* scheme. Specifically, values are additionally indexed by the dates of the events they contain. During retrieval, an LLM $\mathcal{M}_T$ extracts a time range for time-sensitive queries, which filters out a large number of irrelevant values.

Table 2.5: Retrieval performance on the temporal reasoning subset of LongMemEval$_M$. Time-aware query expansion significantly facilitates retrieval by narrowing its scope.

Key Setting	Value = Session				Value = Round			
	Metric@5		Metric@10		Metric@5		Metric@10	
	Recall	NDCG	Recall	NDCG	Recall	NDCG	Recall	NDCG
(1) K = V	0.639	0.630	0.651	0.707	0.421	0.462	0.499	0.511
(1) w/ Query Expansion ($\mathcal{M}_T$ = GPT-4o)	0.654	0.660	0.707	0.679	0.451	**0.565**	0.495	0.538
(1) w/ Query Expansion ($\mathcal{M}_T$ = Llama 3.1 8B Instruct)	0.624	0.627	0.647	0.692	0.384	0.448	0.489	0.488
(2) K = V + fact	0.684	0.688	0.721	**0.782**	0.489	0.500	0.550	0.598
(2) w/ Query Expansion ($\mathcal{M}_T$ = GPT-4o)	**0.722**	**0.732**	**0.797**	0.758	**0.526**	0.548	**0.722**	**0.669**
(2) w/ Query Expansion ($\mathcal{M}_T$ = Llama 3.1 8B Instruct)	0.677	0.688	0.711	0.744	0.481	0.532	0.570	0.647

As shown in Table 2.5, this simple design improves recall by an average of 11.3% when using rounds as the value and by 6.8% when using sessions as the value. This improvement remains consistent when key expansion is applied during indexing. We also find that the effectiveness of this method depends on using a strong LLM for $\mathcal{M}_T$ to accurately infer time ranges from queries. Llama 8B, on the other hand, struggles to generate accurate time ranges, often hallucinating or missing temporal cues even with numerous in-context examples. Further analysis is provided in Appendix E.4 of the original paper [WWY25].

2.4.5 Improving reading with chain-of-note and structured format

As LongMemEval requires syntheses across multiple sessions, even an optimal memory retrieval mechanism needs to return a long context to capture all relevant information. To enhance the model's ability to handle long retrieved contexts, we apply two key optimizations. First, we present retrieved items in a structured JSON format [YVL23], which helps the model clearly recognize memory items as the data for reading. Additionally, we apply the Chain-of-Note (CoN) reading approach [YZP23], instructing the LLM to first extract information from each memory item and then reason based on these notes. This effectively decomposes long-context reading into two simpler subtasks: copying important details and reasoning with more concise notes.

In Figure 2.7, we evaluate the reading designs under the oracle retrieval setting, where only evidence sessions are provided. Surprisingly, even with perfect retrieval, a suboptimal

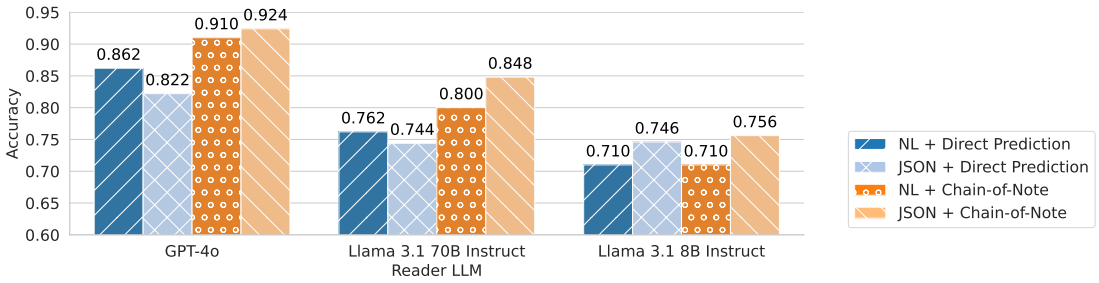


Figure 2.7: Question answering performance under the oracle retrieval setting. CoN with JSON format outperforms the other three parameter combinations by a large margin. NL = Natural Language.

reading strategy results in up to a 10-point absolute performance drop compared to the best approach for GPT-4o. Notably, when CoN is not applied, JSON format does not consistently outperform the natural language format. However, with CoN, JSON format consistently benefits reader LLMs of various capabilities. Section 2.4.6 further analyzes error patterns of LLMs with the enhanced reading strategy.

2.4.6 Error Analysis

In this section, we analyze the source of error of various LLMs with our best memory design. Specifically, we use round as the value granularity, expand the key with extracted user facts, and apply CoN as the reading strategy. Stella v5 1.5B is used as the retriever, and top-10 items are provided to the model with a JSON structured prompt. In Figure 2.8, we analyze the distribution of the correct and error cases for three different reader LLMs. First, we observe that a substantial proportion of errors corresponds to correct retrieval yet wrong generation (15%∼19% of all instances, and 40%∼50% among the error instances). The proportion is higher when a weaker reader LLM is used. This indicates that the reading strategy still has a large room of improvement. In addition, we observe that for the reader LLM to generate a correct answer, performing correct retrieval is necessary in ∼90% of the time. We observe that the rest 10% instances are mostly of the question type knowledge-update and the retriever was only able to identify the updated knowledge but failed to retrieved the

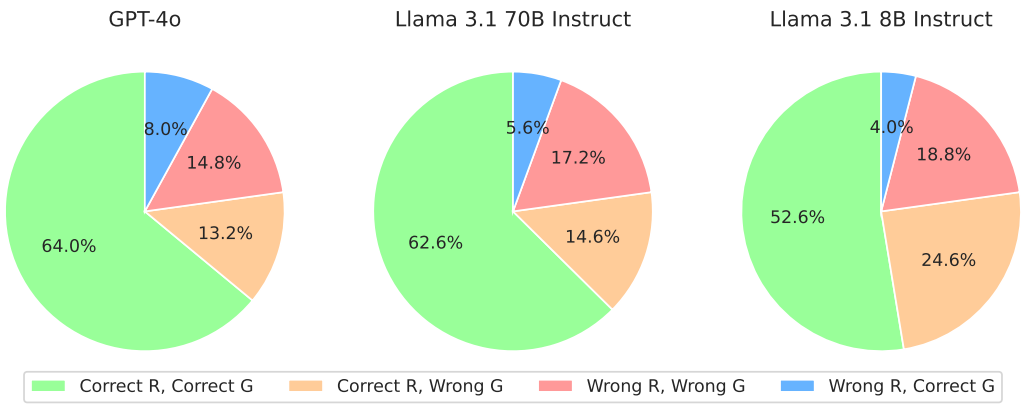


Figure 2.8: An analysis of the error distribution of different reader models. We use R and G to indicate correct Recall@10 and correct answer based on the top-10 retrieved results.

previous information before update. For these cases, our strict retrieval evaluation criteria will deem that the retrieval has failed. Overall, this result also highlights the high quality of LongMemEval, as the reader LLM cannot take any shortcut to answer the question correctly without a correct memory recall.

2.5 Conclusion

In this chapter, we introduced LongMemEval, a comprehensive and challenging benchmark designed to evaluate the long-term memory abilities of chat assistants across five core memory tasks: information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention. Through extensive experiments with both commercial systems and long-context LLMs, we demonstrated the significant challenges posed by LongMemEval, with current systems exhibiting substantial performance drops. By analyzing key design choices across indexing, retrieval, and reading stages, we proposed effective strategies such as session decomposition, fact-augmented key expansion, and time-aware query expansion, which collectively improved both memory recall and the question answering performance. Our findings highlight the need for more sophisticated memory mechanisms to achieve per-

sonalized and reliable conversational AI, and LongMemEval offers a valuable benchmark to drive future advancements in long-term memory capabilities for chat assistants.

Privacy and Trustworthiness LongMemEval and the studied memory system could induce several societal impacts. Storing and recalling user information could cause personal information leakage, and the lack of a memory “deletion" operator could harm the system's trustworthiness. It is also possible that the memory mechanism could be leveraged by malicious parties to toxic contents into the datastore, which may cause a jailbreak behavior during inference. To mitigate such behaviors, it is essential to implement moderation mechanisms that monitor the read/write data flow of the memory. We encourage producers of memory-augmented assistant systems to be aware of these potential harmful effects and apply thorough testing and mitigation.

CHAPTER 3

Memory for Experienced Agents in Specialized Environments

Long-term memory helps large language models (LLMs) operate beyond their context and parameters by storing and recalling information over long horizons [WRH22, PFP23, WGC24]. We now extend the study of memory from personal conversations to agent systems, where LLMs interact with specialized environments over many steps. Recent works show that memorizing task procedures, interface affordances, and hidden failure modes improve agent performance at inference time [WXJ24, ZHX24, BDP25, WMF25, THY25].

However, benchmarks for memory in the agentic context remain limited. Existing memory works mainly evaluate retrieval and reasoning over long documents or user chat histories [HSK24, BTZ25, WWY25, MLT24, TSY25]. Recent works consider evaluating memorization over agent trajectories, but often use simplified game environments [FWL26, LZL26], emphasize limited dependencies within one or a few trajectories [HWZ26, ZYH26], or evaluate indirectly through downstream task success [HWZ26]. As a result, they provide limited insight into whether memory systems can accumulate holistic, environment-specific knowledge from sustained interaction with a complex environment. To highlight this perspective, this paper uses the following framing:

A high-quality memory makes an agent
an experienced colleague in a specialized environment.

Driven by this view, we introduce LongMemEval-V2 (LME-V2), a benchmark for eval-

uating whether memory systems can help web agents acquire the experience needed to become knowledgeable colleagues. LME-V2 leverages customized websites including Magento shopping, shopping admin, Postmill forum, and ServiceNow from WebArena [ZXZ24] and WorkArena [DGC24, BTG24]. From task-solving web agent trajectories, we manually curate 451 questions covering five core memory abilities: static state recall, dynamic state tracking, workflow knowledge, environment gotchas, and premise awareness. We provide examples in Figure 3.1 and ability definitions in §3.2.1. These questions are specific to the customized environments and thus remain generally unanswerable by recent frontier LLMs (§3.2.5). LME-V2 further pairs the questions with a sequence of web agent trajectories ("haystacks", following Kamradt [Kam23]), where only a small fraction bears the answers to each question ("needles"). LME-V2-Small provides a 100-trajectory haystack shared by all questions, and LME-V2-Medium has 500-trajectory question-specific haystacks. Compared to prior benchmarks, LME-V2 poses new challenges with its deep context (25M/115M tokens in the small/medium tiers) and comprehensive memory ability coverage (Table 3.1).

LME-V2 evaluates memory with a *context gathering* formulation (§3.2.4). A memory system implements two APIs: `Insert`, which consumes a trajectory, and `Query`, which returns a multimodal memory context for a question. For each question, we stream the associated trajectories sequentially into memory, invoke `Query`, truncate the returned context to a fixed token budget, and ask a fixed reader LLM to answer. This provides a direct evaluation of memory quality with a practical interface that a downstream agent would use. We report both answer accuracy and query latency.

To succeed in LME-V2, a memory system needs to intelligently store and filter information from the noisy agent trajectories, retaining both low-level observations as well as higher-level environment dynamics and procedural knowledge. As a result, naive application of popular agent memory methods could be ineffective as they are biased towards less noisy conversational contexts [CKA25] or high-level strategic knowledge [WMF25, OYH25]. In this chapter, we propose AgentRunbook, a simple yet effective baseline consisting of two

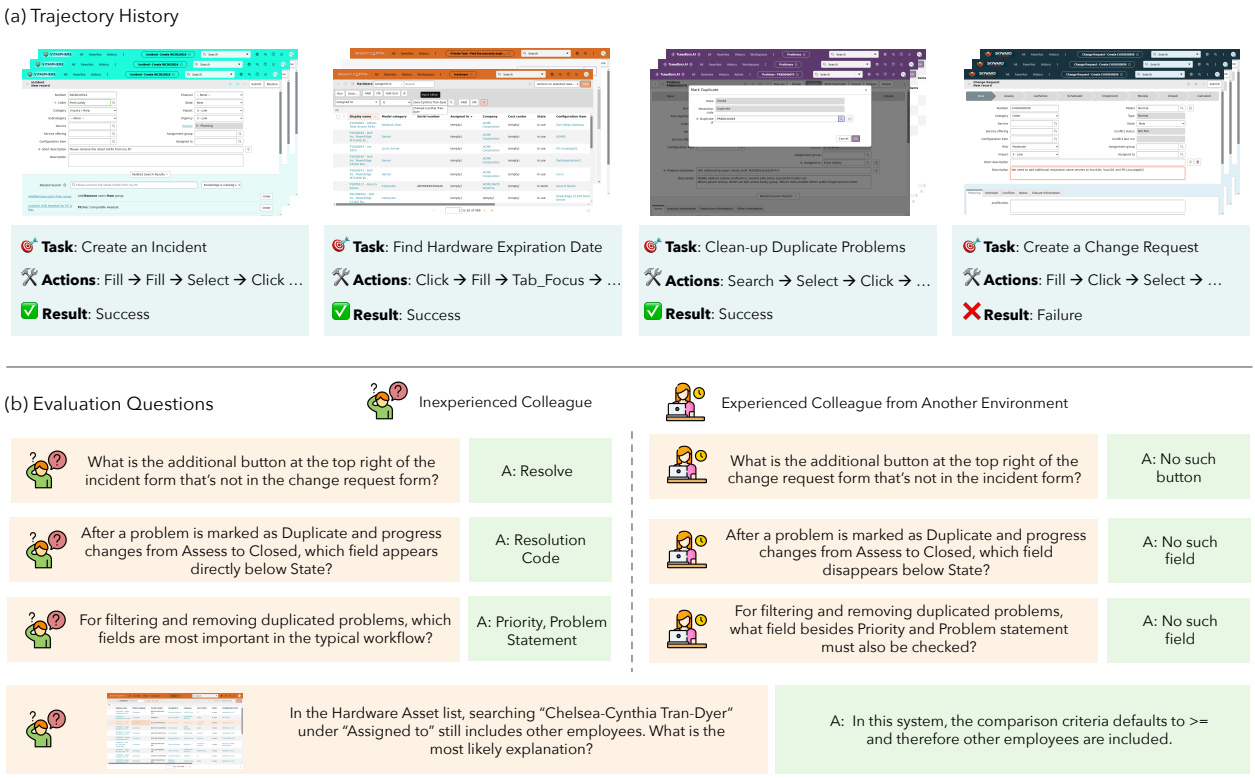


Figure 3.1: Examples of LongMemEval-V2 questions. We display examples from the WorkArena domain. LME-V2 questions exercise diverse memory abilities: static (row 1), dynamic (row 2), workflow (row 3), gotchas (row 4), and premise awareness (right column).

variants, optimized separately for efficiency and accuracy. AgentRunbook-R is an efficient retrieval-augmented generation (RAG) pipeline inspired by agentic memory works such as Xu et al. [XLM25]. It prompts an LLM controller to update and to actively query three knowledge pools: raw observations, state transition events, and high-level strategy notes (§3.3.1). AgentRunbook-R is efficient and covers major memory abilities, but its simple design is not optimized for detailed evidence selection. Inspired by Cao et al. [CYD26], we propose AgentRunbook-C, a coding agent-based memory method that casts memory management as a file management problem. AgentRunbook-C stores raw trajectories directly as files. At query time, it augments an off-the-shelf coding agent harness with workflow documents, memory manifests, and helper scripts, then invokes the agent to assemble a compact evidence set (§3.3.2).

Table 3.1: Comparison with existing memory and long-context benchmarks. # Sess., # Tok., and # Q denote the history size in sessions or tokens, as well as the total number of questions. MM denotes whether the context or question is a multimodal mixture of text and images. For benchmarks with multiple preset length tiers, we report a range between the minimum length tier and the maximum length tier. For the other benchmarks, we report the averaged context size over all examples.

Benchmark	**Domain**	**Context Profile**			**Question Profile**		**Memory Ability**				
		# Sess.	# Tok.	MM	# Q	MM	Static	Dynamic	Workflows	Gotchas	Premise
General Long Context											
LongBench V2	Mixed	N/A	260k	✗	503	✗	✓	✗	✗	✗	✗
MemoryAgentBench	Mixed	N/A	285k	✗	2,071	✗	✓	✗	✗	✗	✗
CL-Bench	Mixed	N/A	10k	✗	1,899	✗	✓	✗	✓	✗	✗
Conversational Long Context											
LoCoMo	User-user chat	28	∼16k	✓	7,512	✗	✓	✓	✗	✗	✓
LongMemEval-V1	User-assistant chat	48–475	115k–1.5M	✗	500	✗	✓	✓	✗	✗	✓
PersonaMem	User-assistant chat	5–60	26k–951k	✗	5,990	✗	✓	✓	✗	✗	✗
PersonaMem-v2	User-assistant chat	10–20	33k–124k	✓	5,000	✗	✓	✓	✗	✗	✗
BEAM	User-assistant chat	4.5–100	124k–10M	✗	2,000	✗	✓	✓	✗	✗	✓
Agentic Long Context											
MemoryArena	Agent (mixed)	7	40k+	✗	766	✗	✓	✗	✓	✓	✗
AgentLongBench	Game agent	1	31k–4M	✗	6,400	✗	✓	✓	✗	✗	✗
EMemBench	Game agent	1	2k–∞	✓	1,280+	✗	✓	✗	✓	✓	✓
FileGramBench	File-system agent	12	11k	✓	4,333	✗	✓	✗	✓	✗	✗
AMA-Bench	Agent (mixed)	1	57k	✗	2,496	✗	✓	✓	✓	✓	✗
LongMemEval-V2	**Web agent**	**100–498**	**25M–115M**	✓	**451**	✓	✓	✓	✓	✓	✓

We evaluate the memory designs on the small and medium tiers of LME-V2. A simple RAG method that retrieves state slices achieves an overall accuracy of 40.1%, and AgentRunbook-R further improves to 57.8%. Accuracy-wise, we find the off-the-shelf Codex agent [Ope] has competitive performance, achieving a surprisingly high 69.3% accuracy. However, the agent achieves this at a cost of about 182 seconds per query, about 6.9 times slower than AgentRunbook-R. With our specialization designs, AgentRunbook-C performs best overall with 72.5% accuracy while being 32% faster than Codex at query time. Our analyses show that AgentRunbook-C significantly advances the accuracy-latency frontier, but the room for future improvement remains large (§3.4.2). Overall, LME-V2 sets a new standard for agent memory evaluation and provides a testbed for memory modules that make long-running agents more reliable, adaptive, and useful in real-world environments.

3.1 Related Work

Long-Context and Memory Evaluation Long-term memory evaluation can be seen as part of the long-lasting effort to evaluate LLMs and retrieval systems on recalling information across extended context. Early line of benchmarks focuses on testing information retrieval, aggregation, and instruction following over long input documents [HSK24, KTL24, MDD25, BTZ25, DZY26]. Subsequent work expanded the focus to personalized memory, covering explicit user facts and implicit preferences, with benchmarks such as LoCoMo [MLT24], DialSim [KCH24], PerLTQA [DWZ24], LongMemEval [WWY25], PersonaMem [JHC25, JYS25], and BEAM [TSY25]. LMEB [ZHX26] isolates the retrieval component and evaluates dense retrievers on memory workloads. In contrast, among a new series of efforts, LME-V2 targets experience memory with context constructed from web agent history trajectories. This shift introduces substantially more complex contexts, a new ability taxonomy, and memory designs centered on agent experience.

Memory Systems for Agents As LLM agents tackle long-horizon tasks in complex environments, memory becomes important both for recalling earlier detailed trajectory context [HCC25] and for consolidating high-level knowledge across trajectories [WMF25, OYH25, WZY25]. Memory has also been linked to improving inference-time performance through extended exploration and sleep-time offline consolidation [OYH25, LSW25]. Despite this progress, direct evaluation of memory quality in agent settings remains limited. MemoryArena [HWZ26] measures memory indirectly through the success rate of interdependent task sequences. AgentLongBench [FWL26] and EMemBench [LZL26] use synthetic agent histories and test recall of details from those traces. FileGram [LTH26] studies reasoning over file system behavior traces. AMA-Bench [ZYH26] is closest to our setting, as it curates questions from agent trajectories in diverse domains such as embodied, web, and gaming agents. However, AMA-Bench focuses on understanding one trajectory, while LME-V2 focuses on environment knowledge induced across many past trajectories. To our knowledge,

LME-V2 is also the first benchmark in this setting to scale the history length to tens or even over 100 million tokens.

Agents as Memory Controllers Recent work on agentic memory proposes memory systems in which memory write and read operations are controlled by an LLM rather than a fixed pipeline. MemGPT [PFP23] and StateLM [LLM26] enable models to manage context programmatically. A-MEM [XLM25] and Mem0 [CKA25] introduce scaffolding that allows an LLM to evolve memory content and structure over time. Memory-R1 [YYH25] and Mem-α [WTL25] learn memory update actions via reinforcement learning. MemSkill [ZLB26] learns memory skills to guide memory update behavior at a finer granularity. In this work, we further expand the notion of agentic memory. Inspired by Cao et al. [CYD26] and Team et al. [THZ26], we view a general coding agent with tool use and file system manipulation abilities as a strong controller for file-based memory. Based on this perspective, we design AgentRunbook-C, which augments an off-the-shelf coding agent harness with workflow documents, query-time rendered artifacts, and helper scripts, yielding a strong accuracy-latency trade-off on LME-V2.

3.2 LongMemEval-V2

3.2.1 Core Memory Ability Definition

What does an experienced colleague internalize after repeatedly working in an environment? We categorize the learned experience into five memory abilities:

- **Static State Recall**. An experienced colleague remembers important landmarks, page layouts, module affordances, and subtle differences across states.
- **Dynamic State Tracking**. An experienced colleague can act as a world model of the environment: given states and actions, they understand how the environment changes.

- **Workflow Knowledge**. An experienced colleague knows the steps needed to perform common tasks in the customized environment.
- **Environment Gotchas**. An experienced colleague is aware of common recurring issues in the current environment and can avoid environment-specific failures.
- **Premise Awareness**. An experienced colleague can recognize assumptions that are valid in another environment but wrong in the current one.

3.2.2 Annotation

To holistically evaluate these memory abilities, we curate LongMemEval-V2 from multimodal web agent trajectories. The annotation has four steps: trajectory collection, question annotation, answer trajectory labeling, and haystack creation. We present full details in Appendix A of the original paper [WJK26].

Trajectory Collection We collect trajectories from three web agent benchmarks: WebArena [ZXZ24], WorkArena [DGC24], and WorkArena++ [BTG24], leveraging their **OneStopShop**, **CMS**, **Reddit**, **ServiceNow** environments. The trajectories are collected using the AgentLab[1] library, which provides unified state representations, action spaces, and a ReAct-style base agent implementation [YZY23]. Using the base agent and Codex [Ope], we perform rejection sampling with GPT-5.2 [Ope25b] and GPT-5-mini [Ope26] as the LLMs. The final pool contains 599 trajectories from WebArena and 941 from WorkArena/WorkArena++. The overall success rate is 52.0%, and each trajectory contains 28.1 states on average.

Question Annotation All questions are constructed through **manual annotation**. Following the memory ability taxonomy, human experts first inspect the trajectories to identify

[1] `https://github.com/ServiceNow/AgentLab`.

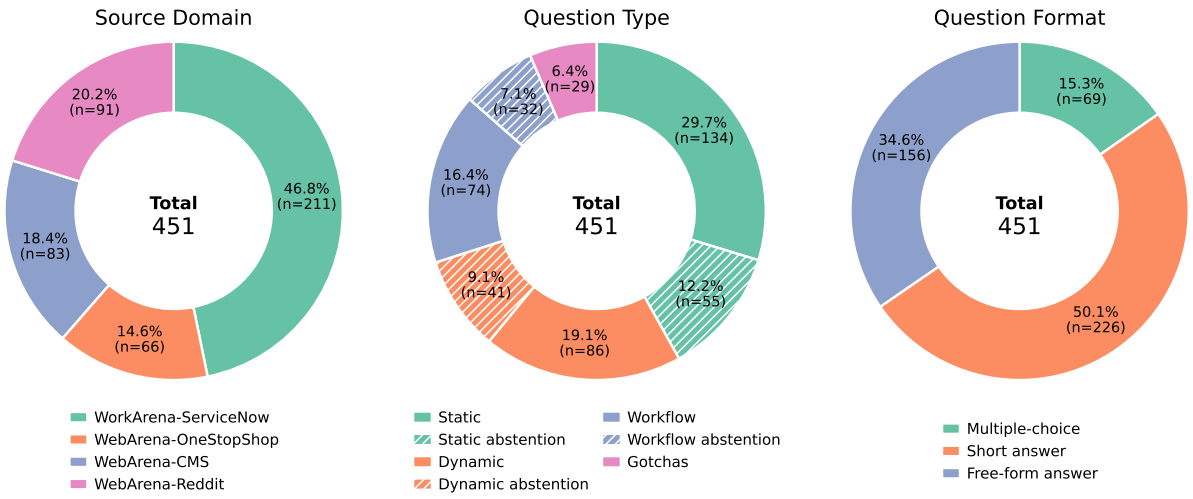


Figure 3.2: LME-V2 questions cover diverse source domains, question types, and formats.

various information an experienced colleague would naturally learn. We then curate and filter questions to ensure strong proprietary LLMs cannot answer from parametric knowledge alone[2]. Gotchas questions are framed as scenarios where an inexperienced worker sends a message with a screenshot, while the other questions are expressed as text-only true/false, multiple choice, or short answer questions. Finally, based on existing static, dynamic, and workflow questions, we curate abstention questions with wrong premises that the model must identify to succeed. Figure 3.1 shows example questions in each category. Figure 3.2 presents source domain, type, and format distribution of the final question pool. On average, questions require 1.4 trajectories to answer (min 1, max 5). However, many dynamic and workflow questions require synthesizing evidence across a supporting trajectory's states.

Answer Trajectory Labeling During annotation, annotators identify a seed set of answer-bearing trajectories for each question. To construct shared history haystacks where we can jointly minimize the number of answer-bearing trajectories for all questions, we perform additional annotation to label all trajectories that contain the answer for each

[2]We manually tested Gemini-3-Pro [Goo25], GPT-5.2 [Ope25b], Grok-4.1-thinking [xAI25], and Claude-Opus-4.6 [Ant26] and ensured that at least two out of four models answered the questions incorrectly.

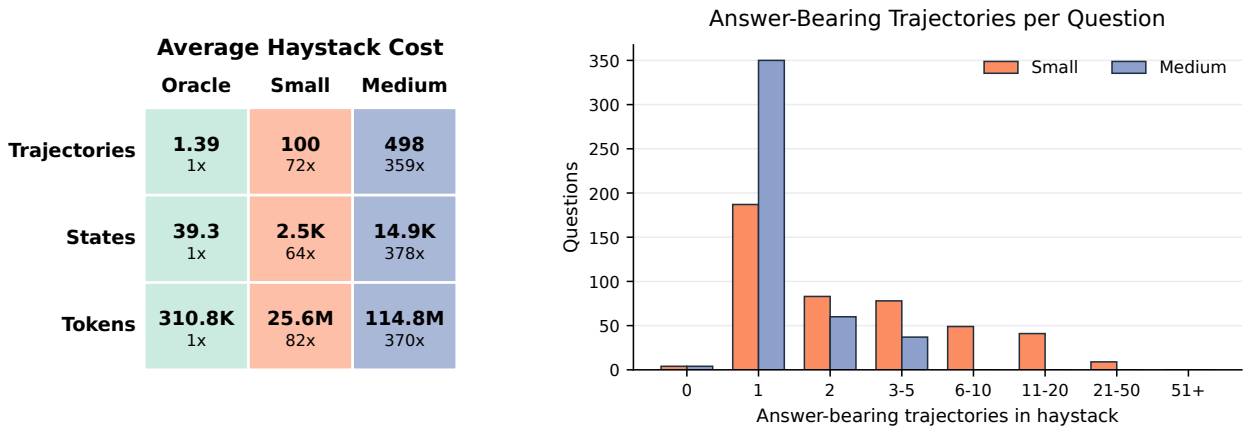


Figure 3.3: Basic statistics of history haystacks in LME-V2. The small and medium tiers cover a large number of states and tokens (left). Meanwhile, the trajectories containing answers to each question remain generally sparse in the haystack (right).

question. We use the Codex coding agent to generate initial proposals. Human experts then verify that the question-trajectory correspondence for trajectories included in the final core haystack set. We provide details in Appendix A of the original paper [WJK26].

Haystack Creation Based on the answer trajectory labels, we programmatically assemble two tiers of history trajectory haystacks: a small variant that contains 100 trajectories shared by all questions, and a medium variant that contains roughly 500 trajectories per question. We refer to them as LME-V2-Small and LME-V2-Medium for the rest of the paper. For LME-V2-Small, we create one haystack for the ServiceNow questions and one haystack for the WebArena domains. All haystacks contain a balanced ratio of successful and failed trajectories, and many questions can only be answered from failed trajectories. Figure 3.3 presents further statistics of the haystacks. The final history lengths of LME-V2-Small and LME-V2-Medium are approximately 25M and 115M tokens, while each question's answer-bearing trajectory set remains sparse in the haystack.

Table 3.1 compares LME-V2 with previous long-term memory benchmarks. LME-V2 has substantially longer histories than prior long-term memory benchmarks, naturally includes multimodal evaluation, and provides a broad coverage of crucial agent memory capabilities.

3.2.3 Evidence Integrity and Haystack Controls

Final trajectory artifact and goal sanitization. For each retained trajectory, the final artifact consists of the task goal and an ordered sequence of state-action pairs. Each state contains the screenshot and the accessibility tree, and each action is the high-level BrowserGym action emitted at that step. For WorkArena and WorkArena++ Level-2 and Level-3 tasks, the original goal descriptions sometimes contain detailed navigational hints, which can make procedure questions answerable by reading the route directly from the goal rather than learning it from the trajectory. We therefore sanitize the goal descriptions while preserving the task intent and task-specific payload. We also sanitize repetitions of the original goal in trajectory states, such as the initial task description and later copied task text.

For all questions, annotators verify that the relevant answer evidence is visible in screenshots. Although accessibility trees sometimes expose information more clearly than screenshots, especially for hidden labels or structured fields, we avoid questions whose answer can only be inferred from the accessibility tree and not from the visual trajectory evidence.

Medium haystack. The medium tier is built per question rather than as a single shared haystack. Importantly, it reuses the same per-question answer seed selected by the small-haystack builder, so the answer-bearing core does not change between tiers. For each question, the builder starts from the selected answer trajectories for its required hops and then adds fillers until the target size is reached, using 500 trajectories for the reported medium tier. Fillers are sampled after excluding the full answer-bearing set for that question, ensuring that the expansion does not introduce extra answer trajectories beyond the chosen seed. The filler ranking promotes task-family diversity, low overlap between filler goals and answer-core goals, low global answer-support count, and outcome balance. For zero-support abstention questions, the builder reuses the manually verified anchor question's exact haystack.

3.2.4 Evaluation Formulation

We formulate LME-V2 as a **context gathering** task. For each question q_i with gold answer y_i, a memory system receives an ordered trajectory haystack $\mathcal{H}_i = \{h_{i,1}, \ldots, h_{i,m_i}\}$, where each $h_{i,j}$ is a trajectory. The system must support two APIs, $\texttt{Insert}(h)$ and $\texttt{Query}(q)$. We sequentially insert all trajectories in $\mathcal{H}_i$, query the final memory with q_i, and obtain a returned context c_i:

$$\mathcal{M}_{i,j} = \texttt{Insert}_{\mathcal{M}_{i,j-1}}(h_{i,j}),$$

$$c_i = \texttt{Query}_{\mathcal{M}_{i,m_i}}(q_i).$$

A fixed reader model R answers from the question and a bounded memory context: $\hat{y}_i = R(q_i, \text{Trunc}(c_i))$[3]. We report answer accuracy and query latency. Accuracy is computed by normalized string matching for structured answers and an LLM judge for free-form answers.

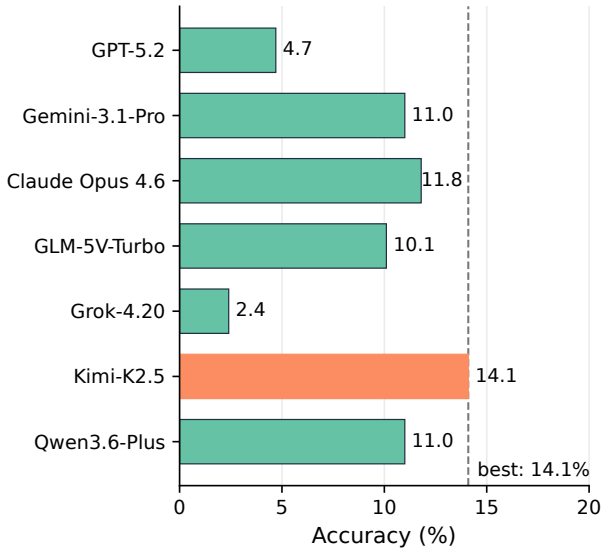
Structured answers are scored with deterministic evaluators specified by each question, including normalized phrase matching, ordered phrase matching, single-choice matching, and multi-select choice matching. Gotchas and abstention questions are evaluated with an LLM judge because they require semantic comparison to the reference insight or flawed-premise explanation. The judge outputs a binary JSON label, which is used as the final score. We aggregate accuracy over the full set and also report category-level breakdowns.

3.2.5 Pilot Studies

We perform two pilot studies. First, we evaluate whether LME-V2 questions require environment-specific trajectory evidence. Then, we sanity check whether answer-bearing trajectories are sufficient for reliable question answering. These studies use a direct question answering setup rather than the context gathering formulation used in the main experiments, and evaluate non-abstention questions only. Full per-category results, prompts, and sandbox instructions are provided in Appendix B of the original paper [WJK26].

[3]We set the truncation budget to 200k tokens empirically.

(a)



(b)

Context	Overall	Static	Dynamic	Workflow	Gotchas
Qwen3.5-9B (thinking enabled)					
No context	0.016	0.000	0.015	0.155	0.136
Oracle trajectories	0.596	0.566	0.668	0.718	0.310
Oracle slices + notes	0.825	0.908	0.879	0.750	0.484
GPT-5.4-mini (medium reasoning)					
No context	0.045	0.025	0.010	0.075	0.171
Oracle trajectories	0.653	0.660	0.696	0.697	0.484
Oracle slices + notes	0.863	0.950	0.843	**0.905**	0.467
Codex + GPT-5.4-mini (xhigh reasoning)					
Oracle trajectory files	**0.897**	**0.986**	**0.947**	0.815	**0.517**

Figure 3.4: Pilot studies on LME-V2 non-abstention questions. (a) Frontier LLMs perform poorly without trajectory history, suggesting that parametric knowledge alone is insufficient for LME-V2. (b) LME-V2 is challenging to answer even with oracle answer-bearing trajectories and optimizations such as evidence slicing with notes or using a coding agent harness help improve performance.

Can recent frontier LLMs answer LME-V2 questions without trajectory history? We prompt strong LLMs with only the question. As shown in Figure 3.4 (a), all LLMs perform poorly in this setting: the best model reaches only 14.1% overall accuracy, suggesting that LME-V2 questions generally cannot be answered from public or parametric knowledge alone.

Second, we give models oracle access to the answer-bearing trajectories to isolate the difficulty of reading and grounding trajectory evidence. Long-context prompting shows much higher accuracy but remains limited due to the trajectory size exceeding the model's context window. We further consider two techniques: 1) annotating ground-truth states containing the evidence and providing only radius-1 evidence slices around them and 2) summarizing strategy notes containing important procedures and gotchas identified in the trajectory. These two techniques further improve direct QA to 82.5% and 86.3%, respectively. Finally, we represent the trajectories as files and use the off-the-shelf Codex coding agent to directly answer the question. Surprisingly, GPT-5.4-mini with the Codex harness answers the questions better than prompting approach, suggesting that detailed evidence inspection via multi-step tool use is effective for understanding agent trajectories, and that coding agents might have good performance acting as memory controllers. Overall, these findings confirm that the answer trajectory labels are accurate enough and motivate our memory design.

3.3 AgentRunbook

LME-V2 is challenging because the evidence needed for a question can mix low-level UI observations, state transitions, and reusable task procedures. Memory modules therefore need to organize noisy agent trajectories into compact representations and index them for targeted recall. We propose two memory designs: AgentRunbook-R, a structured RAG pipeline with separate knowledge pools, and AgentRunbook-C, a coding agent based method that casts memorizing agentic contexts as a file management problem. Figure 3.5 illustrates the workflow of both methods.

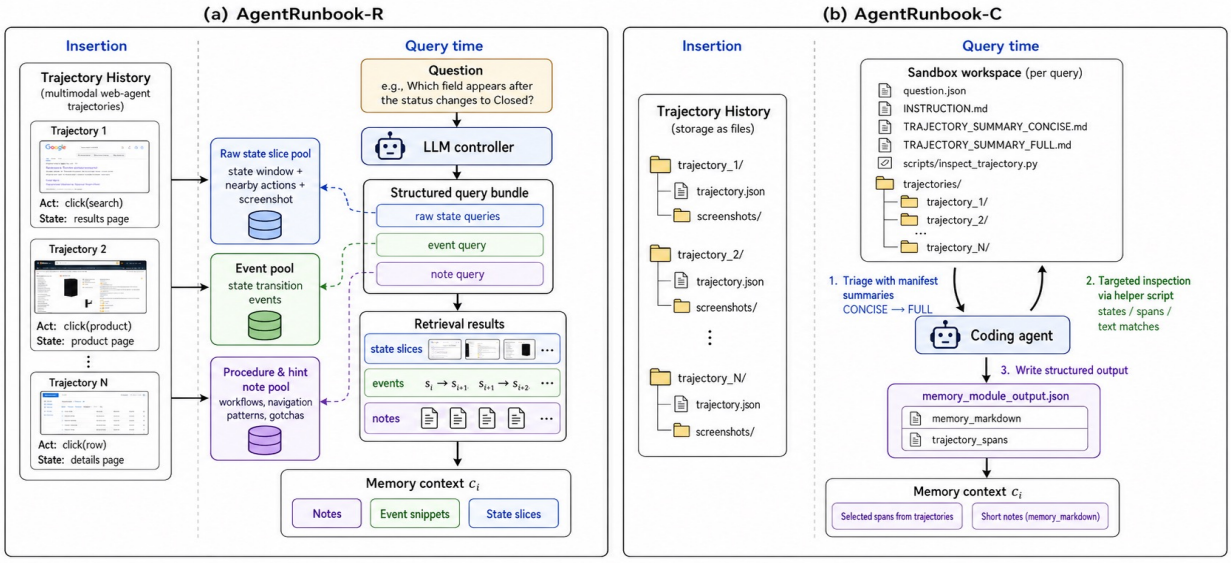


Figure 3.5: An illustration of the AgentRunbook memory modules. (a) AgentRunbook-R digests trajectories at insertion time and creates dedicated knowledge pools for raw states, events, and notes. At query time, an LLM controller generates multi-stream queries for each pool. (b) AgentRunbook-C stores trajectories as files at insertion time and constructs a sandbox workspace with specialized instructions and manifests for a coding agent to collect the required evidence efficiently.

3.3.1 AgentRunbook-R

AgentRunbook-R, where R denotes RAG, extracts structured memory items at insertion time and retrieves them at query time. To recall information at different granularities, AgentRunbook-R uses separate knowledge pools and a retrieval mechanism over these pools. Given a trajectory h, AgentRunbook-R builds three memory pools. The **raw state slice** pool stores windows centered at trajectory states, including local UI observations and nearby actions. This pool preserves fine-grained visual and textual evidence. The **state transition event** pool stores events extracted from consecutive states. These events describe how actions change the environment, accumulating evidence for an environment world model. The **procedure and hint note** pool stores trajectory-level notes that capture reusable workflows, navigation patterns, and environment-specific gotchas. This pool draws on work that consolidates trajectory experience into compact reusable knowledge [WMF25, OYH25].

At query time, AgentRunbook-R uses an LLM controller to reason about the query and the current memory snapshot, then generate retrieval queries for the knowledge pools: multiple raw state queries for exact UI evidence, one event query for important state changes, and one note query for procedural knowledge. The controller may skip irrelevant streams. Each query retrieves from its corresponding pool using dense retrieval, and the results are rendered as a multimodal memory context c_i. This design keeps query latency low while supporting evidence at different granularities.

3.3.2 AgentRunbook-C

AgentRunbook-C, where C denotes coding agent, is motivated by the observation that general coding agents are effective file system manipulators and tool users [CYD26]. Rather than compressing retrieval behavior into a fixed vector search pipeline, AgentRunbook-C stores trajectories directly as files and uses a coding agent to search, inspect, and select evidence at query time.

An off-the-shelf coding agent, however, is not optimized as a memory module. It may over-explore, under-explore, or inspect the data inefficiently. AgentRunbook-C adds three lightweight scaffolding components to the coding agent. First, a **workflow document** instructs the agent to act as a memory module and outlines the steps for collecting evidence. Second, query-time **manifest artifacts** summarize the current memory layout, helping the agent shortlist relevant trajectories before detailed inspection. Third, a **helper script** exposes common trajectory inspection operations, such as viewing a state span or searching within a trajectory. At insertion time, AgentRunbook-C stores each trajectory on disk. At query time, it creates a sandbox with the question, workflow document, helper script, and rendered manifest artifacts. The coding agent writes a structured retrieval output containing a short memory note and selected trajectory state spans, rendered into c_i.

The `memory_markdown` field contains brief support analysis and any relevant procedure or hint notes. The trajectory spans are zero-based inclusive state ranges, with a total span

budget of 20 states. After the coding agent finishes, AgentRunbook-C validates the JSON output, filters invalid spans, renders the selected states and screenshots into the returned memory context, and passes this context to the fixed reader.

3.4 Experiments

We evaluate all methods on LME-V2 under the context gathering formulation. The returned memory context is truncated to 200K tokens and answered by a fixed Qwen3.5-9B reader [Qwe26a]. For RAG methods, we use Qwen3.5-9B as the memory controller and Qwen3-Embedding-8B for retrieval. For coding agent methods, we use Codex and GPT-5.4-mini with different reasoning efforts. The full implementation details of AgentRunbook and baselines are provided in Appendix C of the original paper [WJK26].

3.4.1 Main Results

As shown in Table 3.2, the no-retrieval baseline is near zero, confirming that the reader cannot answer without memory context. A simple query-to-slice RAG baseline reaches 42.8% on LME-V2-Small and 38.1% on LME-V2-Medium, while adding trajectory notes improves performance to 51.0% and 45.9%. AgentRunbook-R further improves over the strongest RAG baseline, reaching 58.6% on LME-V2-Small and 57.0% on LME-V2-Medium. The ablations show that the raw slice pool is important for static questions, while removing the event pool harms static, dynamic, and gotchas questions. Workflow questions benefit from consolidating trajectory experience into reusable events and notes rather than only retrieving local observations.

AgentRunbook-C achieves the best overall accuracy, reaching 74.9% on LME-V2-Small and 70.1% on LME-V2-Medium. It also improves over vanilla Codex, which reaches 69.9% and 68.7%, respectively. The ablations show that workflow instructions are consistently important, while manifest artifacts mainly improve efficiency. Helper functions affect the

Table 3.2: Main results with baselines and ablations. The downstream reader is always Qwen3.5-9B. We boldface the best results in each method family. ✣ means statistically significantly outperforming the non-ablation baselines via paired bootstrap test ($p < 0.05$). AgentRunbook strongly outperforms the baseline in RAG family and achieves a superior latency in coding agent family.

LME-V2-Small						
Method	**Overall**	**Static**	**Dynamic**	**Workflow**	**Gotchas**	**Latency**
No retrieval	0.013	0.000	0.008	0.094	0.138	0s
RAG Methods (Controller = Qwen3.5 9B, thinking enabled)						
RAG: query → slice	0.428	0.471	0.425	0.415	0.207	0.1s
RAG: query → slice + notes	0.510	0.524	0.496	0.528	0.414	0.2s
AgentRunbook-R	**0.586**✣	**0.661**✣	0.583✣	0.528	0.310	26.9s
– raw slice pool	0.423	0.286	0.551	**0.538**	**0.345**	16.7s
– event pool	0.556	0.614	0.559	0.528	0.276	19.1s
– note pool	0.579	0.651	**0.614**	0.481	0.310	22.8s
Coding Agent Methods (Controller = GPT-5.4-mini, xhigh reasoning)						
Codex	0.699	0.804	0.670	0.575	**0.586**	177.2s
AgentRunbook-C	**0.749**✣	0.820✣	**0.724**✣	**0.726**✣	0.483	108.3s
– workflow	0.701	0.772	0.677	0.632	**0.586**	167.9s
– manifest artifacts	0.747	**0.847**	0.709	0.698	0.448	155.0s
– helper functions	0.714	0.783	**0.724**	0.660	0.414	145.9s

LME-V2-Medium						
Method	**Overall**	**Static**	**Dynamic**	**Workflow**	**Gotchas**	**Latency**
No retrieval	0.013	0.000	0.008	0.094	0.138	0s
RAG Methods (Controller = Qwen3.5 9B, thinking enabled)						
RAG: query → slice	0.381	0.434	0.405	0.293	0.242	0.1s
RAG: query → slice + notes	0.459	0.487	0.472	0.434	0.310	0.3s
AgentRunbook-R	**0.570**✣	**0.630**✣	**0.614**✣	**0.472**✣	**0.345**	25.8s
– raw slice pool	0.335	0.233	0.433	0.377	0.413	17.1s
– event pool	0.484	0.534	0.496	0.434	0.276	18.5s
– note pool	0.499	0.561	0.543	0.396	0.276	20.5s
Coding Agent Methods (Controller = GPT-5.4-mini, xhigh reasoning)						
Codex	0.687	0.783	0.646	0.613	**0.517**	185.8s
AgentRunbook-C	0.701	0.788	**0.701**✣	0.613	0.449	139.9s
– workflow	0.641	0.709	0.646	0.575	0.414	231.9s
– manifest artifacts	0.681	0.767	0.685	0.576	0.483	211.6s
– helper functions	**0.718**	**0.804**	0.693	**0.689**	0.380	182.5s



Figure 3.6: AgentRunbook improves the query accuracy-latency frontier of memory modules. The panels compare LME-V2-Small and LME-V2-Medium. Each operating point shows answer accuracy and query latency under the labeled reasoning effort setting. AgentRunbook-R supports efficient querying, while AgentRunbook-C improves the trade-off over the Codex baseline.

performance in a mixed way: they improve the small-tier result and reduce latency compared with the most expensive ablations, but their effect on medium-tier accuracy is not uniformly positive. Overall, the results show that coding agents can serve as strong memory controllers with a proper file-based environment. In Sections 3.4.5 and 3.4.6, we further analyze the error patterns and the tool calling behavior of AgentRunbook-C.

3.4.2 Accuracy and Latency Trade-off

An ideal memory method should support both accurate and efficient querying. Across multiple factors, we find the reasoning effort of the memory controller has a large and direct effects on the overall query latency. We thus use it to analyze the methods across different operating points. As shown in Figure 3.6, AgentRunbook-R provides a moderate-accuracy, low-latency baseline: it substantially improves over the slice-plus-note baseline while keeping latency around 26 seconds, or much lower without thinking. This makes it a strong choice when query efficiency is prioritized. AgentRunbook-C moves the accuracy and latency frontier upward. Across reasoning effort settings, the scaffolded coding agent memory

consistently offers a better trade-off than directly using the off-the-shelf coding agent. This suggests that coding agents are more effective as memory controllers when paired with explicit workflow guidance, manifests, and trajectory inspection tools.

3.4.3 LLM Judge Validation

The evaluation protocol relies on an LLM judge only for gotcha and abstention questions, covering 156/451 questions (35%). To assess the reliability of this scoring, we meta-evaluated the judge on LME-V2-Small using outputs from RAG (slices + notes), Codex, AgentRunbook-R, and AgentRunbook-C, with two independent human reviewers. Table 3.3 reports the results. The abstention judge is highly precise but conservative; the gotcha judge is more balanced but slightly noisier.

3.4.4 Sensitivity to the Reader Model

The main experiments use a fixed Qwen3.5-9B reader to compare memory systems under the same answering model. To check whether the comparison depends on this choice, we re-evaluated identical contexts from all four systems on LME-V2-Small with GPT-5.4-mini as the downstream reader (Table 3.4). The reader swap preserves the full point-estimate ranking: AgentRunbook-C > Codex > AgentRunbook-R > RAG. GPT also retains 1,033/1,147 (90.1%) Qwen-correct system–question pairs. The strict ranking holds in 80.5% of 100,000 paired question-level bootstrap replicates (451 questions sampled with replacement while keeping systems and readers paired).

3.4.5 Error Analyses

We categorize each incorrect final answer into one mutually exclusive error type. Retrieval errors are defined against the final human labeled answer-trajectory coverage maps. For each question, we take the answer-bearing trajectory set and derive each trajectory's task family,

Table 3.3: LLM-judge validation on LME-V2-Small. Agreement and κ measure consistency between the two human reviewers.

Decision	Precision	Recall	F1	Human agreement	Cohen's κ
Abstention	99.7%	80.9%	89.3%	95.1%	0.902
Gotchas	86.5%	83.1%	84.7%	94.6%	0.892
Overall	—	—	—	95.0%	0.900

Table 3.4: Reader sensitivity on LME-V2-Small using identical memory contexts.

System	Qwen3.5-9B	GPT-5.4-mini	Qwen-correct cases retained by GPT
RAG (slices + notes)	51.0%	54.8%	87.0%
AgentRunbook-R	58.6%	65.0%	87.9%
Codex	69.9%	67.2%	90.8%
AgentRunbook-C v5.3	74.9%	72.3%	93.2%

which contains the tasks with the same goal but slightly different data and starting points. A major retrieval miss means that the returned raw slices or state spans do not hit any answer-bearing task family. A minor retrieval miss means that the returned evidence hits the answer task family, but not an exact answer-bearing trajectory. If the returned context hits an exact answer-bearing trajectory, answer URL, or gold-answer text, but the reader still answers incorrectly, we label the failure as a reading error. We remark that under this categorization, the memory module often still has the responsibility to the reading errors as it has the freedom to use better state slices and better evidence presentation methods. Gotcha and premise/abstention failures are kept separate for better visualization.

As shown in Figure 3.7, AgentRunbook-R significantly reduces retrieval errors and downstream reading errors compared to the RAG+notes baseline. It does not improve abstention as it directly presents the relevant evidence to the downstream model without an analysis and thus the model can be misled into using the evidence to the question instead of rejecting it. Both Codex and AgentRunbook-C make fewer retrieval errors and downstream reader errors compared to the RAG methods. AgentRunbook-C further improves the abstention performance as the memory module is also instructed to explicitly identify the inconsistencies

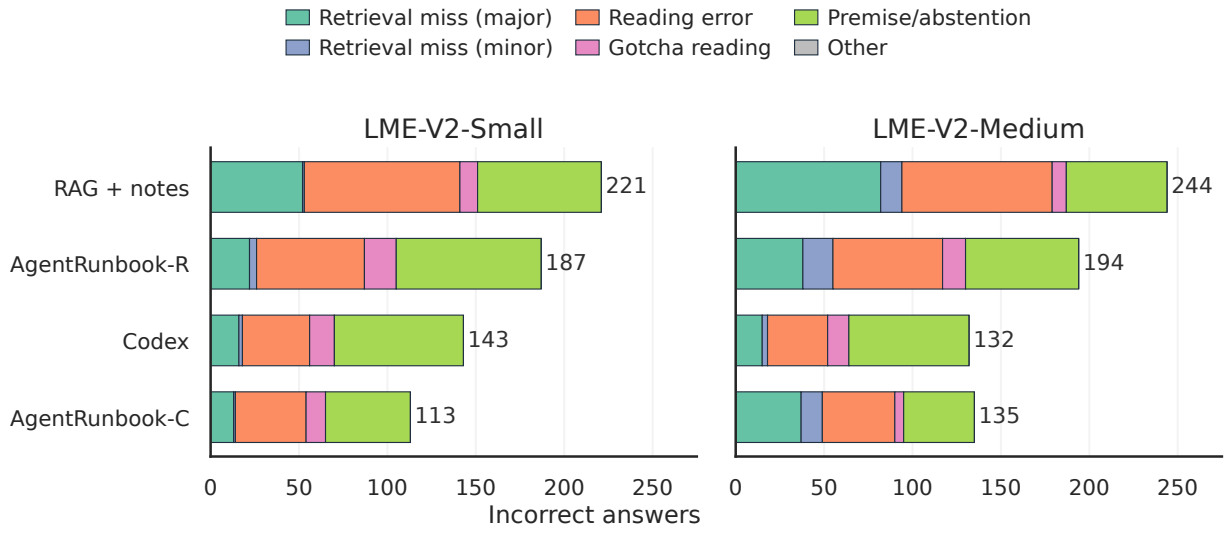


Figure 3.7: Error composition for the main methods on LME-V2-Small and LME-V2-Medium. Each horizontal bar decomposes the incorrect answers for one method into retrieval misses, reading failures, gotcha failures, and premise-awareness failures.

and wrong question premises and present them to the downstream model.

3.4.6 Tool Calling Behavior

We compare AgentRunbook-C with the Codex baseline by parsing the coding agent event streams and grouping shell commands into five behavior classes. The *setup/read task* group includes orientation and prompt/question reads. *Harness-guided retrieval* includes manifest/summary reads and AgentRunbook helper calls. *Raw trajectory exploration* includes broad filesystem searches, direct trajectory reads, and ad-hoc Python scans. The remaining groups capture visual inspection and output validation or other commands.

As shown in Figure 3.8, AgentRunbook-C reduces the total number of commands and shifts work from raw trajectory exploration toward harness-guided retrieval. On LME-V2-Medium, Codex uses 21.8 raw trajectory-exploration commands per question on average, while AgentRunbook-C uses 18.0 harness-guided retrieval commands and only 1.2 raw trajectory exploration commands.

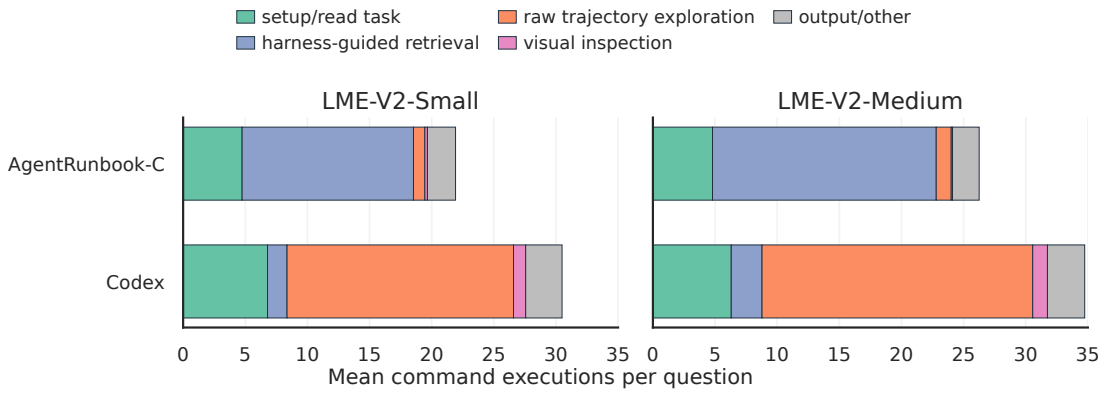


Figure 3.8: Mean number of command executions per query divided by command types. Each stacked bar compares the command use of AgentRunbook-C and Codex on LME-V2-Small or LME-V2-Medium. AgentRunbook-C reduces the total number of commands and shifts work from raw trajectory exploration toward harness-guided retrieval.

3.5 Conclusion

We introduce LongMemEval-V2, a long-term memory benchmark that formulates a new standard for agent memory evaluation: memory systems should help agents become experienced operators of specialized environments. LME-V2 holistically covers five memory abilities and advances the context depth of memory benchmarks with beyond 100M-token context from large multimodal web-agent histories. We further propose AgentRunbook-R, which improves standard RAG-based methods with dedicated memory pools, and AgentRunbook-C, which leverages the file manipulation abilities of coding agents and further improves the accuracy and latency through lightweight workflow guidance, manifests, and inspection tools. We hope LME-V2 provides a concrete testbed for memory modules that make long-running agents more intelligent and reliable in real-world environments.

3.6 Limitations

Benchmark scope and formulation. LME-V2 focuses on web agents operating in customized browser environments. Browser use is a broad and practically important domain,

but it does not cover the full space of digital agents, such as coding agents, computer-use agents, or domain-specific enterprise agents, which may involve different memory requirements and risk profiles. In addition, for reproducibility and controlled comparison, LME-V2 evaluates memory over pre-collected trajectory histories rather than online learning during live task execution. This design makes the benchmark easier to distribute and reproduce, but it may not fully capture distribution shifts caused by an agent's own evolving behavior. Finally, our context gathering formulation evaluates whether a memory module can return useful evidence for a fixed reader model, rather than directly measuring end-to-end task success. This is an intentional design choice to isolate memory quality, but downstream agent correctness may also depend on planning, tool use, and action execution.

Methodological scope. AgentRunbook explores practical memory designs built around retrieval, file organization, and agent-native evidence inspection, rather than proposing new model architectures or training procedures. While this keeps the methods simple and easy to analyze, their performance depends on the quality of trajectory representations, retrieval models, prompts, and coding agent behavior. AgentRunbook-C significantly improves over an off-the-shelf Codex harness in our experiments, but we do not build a fully customized coding agent harness from scratch. Future work could study tighter integration between memory, planning, and execution, as well as learned memory controllers that adapt their storage and retrieval strategies across environments.

Part II

Selective Knowledge Access

CHAPTER 4

Selective Retrieval for Repository-Level Code Completion

Automatic code completion has attracted long-lasting research efforts due to its high practical value in improving programmer productivity [YF02, HR04, HD17]. One particularly challenging scenario is *repository-level code completion*, where a system is required to complete lines, API invocations, or functions in a file from user repositories. For this task, language models for code (code LMs) have emerged as a promising solution due to their ability to leverage the context of the current file to generate coherent code of flexible granularity

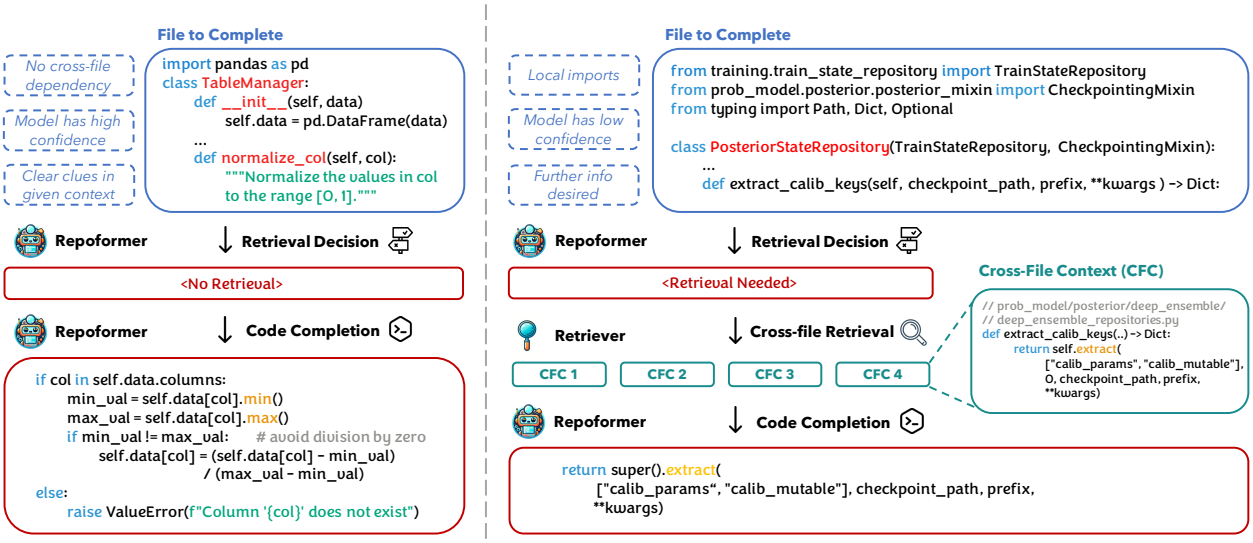


Figure 4.1: An overview of the proposed selective RAG framework. Given the current file context, the system first assesses whether retrieval is required and triggers the retriever if the question can likely be benefited from retrieval (right), abstaining from retrieval otherwise (left). Then, the code LM generates with optional retrieved contexts. With REPOFORMER, the two stages are streamlined via self-assessment.

[TSD14, SDF20, CTJ21]. However, these approaches fail to capture the holistic repository knowledge spanning beyond the current file, such as user-defined APIs and inter-module dependencies [ZCL22, ZCZ23, DWA23]. Recently, the *retrieval-augmented generation* (RAG) paradigm was proposed to bridge the gap: cross-file contexts such as relevant code snippets or documentations are retrieved and provided to code LMs as augmentations to the current file. This approach has shown strong empirical performance and was further advanced by recent literature through designing better retrieval mechanisms for prompting black-box code LMs [LDH22, SLT23, ZCZ23] and adapting the LM to better leverage structured retrieved contexts such as classes, functions, or APIs [DWA24, ZCL22].

Despite their encouraging performance, existing RAG-based approaches largely ignore to address a critical question:

Should we always perform retrieval augmentation?

Our findings suggest that the answer is predominantly negative. First, in various code completion tasks, we discover that up to 80% of the retrievals performed by a standard RAG method *do not enhance the performance* of common code LMs such as CodeGen [NPH23] and StarCoder [LAZ23], and many degrade the performance by introducing irrelevant information (Section 4.4.1). Second, always retrieving introduces notable *inefficiencies*. For moderately sized repositories, sparse retrieval is already as time consuming as code completion with a 3B code LM (Section 4.4.3 and Section 4.5). This inefficiency is more pronounced with dense retrieval, enterprise-scale repositories, and iterative RAG methods such as Zhang et al. [ZCZ23].

In this chapter, we challenge the assumption of always retrieving by proposing a novel repository-level code completion framework underpinned by a *selective retrieval* mechanism: the system proactively abstains from performing unnecessary or potentially detrimental retrievals (Figure 4.1 (a)). At the core of our framework is REPOFORMER, an intelligent code LM fine-tuned for *robust code completion with self-triggered retrieval augmentation*. REPO-

FORMER reflects three core principles:

1. **Performance-oriented self-evaluation.** After observing the current file, REPOFORMER explicitly expresses the likelihood that its prediction quality could be improved by cross-file retrieval. Our training strategy enables the model to combine two factors in this decision: the *code LM* already knowing the answer without retrieval [KCA22] and the *code completion question* not depending on cross-file information and thus retrieval is likely uninformative.
2. **Robustness to retrieved contexts.** REPOFORMER learns to use the retrieved contexts to improve the quality of its output and avoid performance drops caused by potentially noisy retrieved information.
3. **Generalizability.** The aforementioned two abilities must generalize to any completion granularity, programming language, and retriever choice. In addition, REPOFORMER should be able to function as a plug-and-play selective retrieval policy when other models are employed as the generation model.

We posit that these abilities can be faithfully obtained by learning from *simulations of RAG*. Specifically, we leverage a large number of permissively licensed repositories, sample diverse blanks to complete, and pair them with the retrieved repository-level cross-file contexts. Then, for a given code LM, the ground-truth label for selective retrieval is obtained by contrasting the quality of its outputs with and without retrieval augmentation. With this dataset, we design a self-supervised objective to jointly train code LMs to accurately self-evaluate the need for retrieval and robustly complete the code with the optional retrieval augmentation (Section 4.2.3).

We perform comprehensive evaluations on a range of repository-level code completion tasks from RepoEval [ZCZ23], CrossCodeEval [DWA23], and CrossCodeLongEval a new large-scale benchmark focusing on code chunk and function completion. Results show that REPOFORMER achieves strong performance, outperforming always retrieving with the same-sized StarCoderBase by more than 3 absolute points for edit similarity across multiple tasks.

The 3B REPOFORMER performs on par with always retrieving using the 16B StarCoder, and the 16B REPOFORMER achieves state-of-the-art performance across all the tasks (Section 4.4.2). Furthermore, our framework allows for up to 70% inference speedup without harming accuracy. We also establish that REPOFORMER can accelerate RAG with larger black-box LMs as a plug-and-play selective RAG policy, improving the performance while reducing the latency of line and API completion to 75% (Section 4.4.3).

Finally, in Section 4.5, we provide comprehensive analyses on REPOFORMER's generalization ability. We show that REPOFORMER makes precise retrieval abstention decisions, is robust to retrieved contexts, and performs well when tested in other languages or with other retrievers. To facilitate future research on repository-level code completion, we will release our implementation and the CrossCodeLongEval benchmark at `https://repoformer.github.io/`.

4.1 Related Work

Repository-level Code Completion Accurately completing the code in repositories has been a challenging research problem due to cross-file dependency patterns caused by modular design [Par72, TSD14]. Early works propose application-specific training methods for n-gram LMs [TSD14], RNNs [HD17, WSD21], and Transformers [SDF20] to leverage structured knowledge beyond current file's context. Recent studies investigate fine-tuning powerful pre-trained code LMs [CTJ21, NPH23, LAZ23] to better leverage retrieved knowledge provided in context such as code and documentation snippets [ZCL22, DWA24, SKV23]. Concurrently, other studies show that black-box code LMs can already take advantage of in-context knowledge, depending on how well the knowledge is retrieved and formatted [LDH22, ZAX23, SLT23, ZCZ23]. This approach does not require one to train the LM and thus promises better generalization. Orthogonal to these studies, this paper identifies and addresses the robustness and efficiency issues caused by invariably performing the

retrieval augmentation. Our solution takes the form of selective retrieval augmentation through self-assessment.

Adaptive RAG This chapter is consistent with the recent trend of making the RAG paradigm active and adaptive. A core question is finding an effective policy to decide *when to retrieve*. He et al. [HNB21] propose to learn to adjust the importance weight of retrieval based on language modeling performance. Drozdov et al. [DWR22] proposes to upweight the retrieved information when the retrieval has high quality. Li et al. [LTZ23] and Jiang et al. [JXG23] suggest that retrieval should be performed only when LMs have a high predictive uncertainty. Mallen et al. [MAZ23] discover that retrieval can be avoided for popular facts. Concurrent to this work, two new studies approach adaptive RAG from a learning perspective. SKR [WLS23] collects instances where retrieval is not helpful for black-box LMs and proposes several methods to predict these instances. Self-RAG [AWW24] utilizes GPT-4 [Ope23a] as a knowledge engine to distill a smaller LM to evaluate whether answering a question can be benefited from retrieval. In comparison, this paper highlights the importance of understanding whether an LM knows the answer [KCA22] in forming the retrieval policy. We introduce a simple yet effective scheme to fine-tune a code LM for faithful self-evaluation without extra modules (SKR), knowledge store (SKR), or labels generated by an oracle LM (Self-RAG). We show that our approach leads to no performance harms (Section 4.4.2), substantial speedup (Section 4.4.3), and a high decision accuracy (Section 4.5).

4.2 Approach

In this section, we first briefly formulate the repository-level code completion task and the considered RAG setup. Then, we illustrate the details of the proposed framework.

4.2.1 Background

Problem Formulation We denote *repository-level code completion* by (X_l, X_r, Y, F). Y is the ground truth completion that needs to be generated. In this chapter, Y always contains one or more consecutive lines of code. X_l and X_r are the code to the left/right of Y in the same file. We will use the left/right context to refer to them. F is the set of other files in the repository. A code completion system utilizes X_l, X_r, and F to generate a hypothesis $\hat{Y}$.

Retrieval-Augmented Generation We follow the RG-1 formulation in Zhang et al. [ZCZ23] to execute RAG for code completion in four stages: *indexing*, *query formation*, *retrieval*, and *generation*. We consider two components:

- An **in-repository retriever** $\mathcal{R}$ that queries F with information from X_l and X_r and returns relevant cross-file contexts CC. CC consists of k code chunks $cc_1, cc_2, ..., cc_k$, each of which contains consecutive lines of code extracted from a file in F. We mainly use Jaccard similarity [Jac12] as $\mathcal{R}$ due to its speed and strong performance [ZCZ23].
- A **code LM** $\mathcal{M}$ that leverages X_l, X_r, and CC to output $\hat{Y}$. The inclusion of X_r and CC is optional. In this chapter, we always directly provide X_r in the prompt in addition to X_l [SLT23, PZL23]. We provide empirical support for this design in the original paper's appendix [WAZ24].

Full documentation of the RAG stages and their hyperparameters are provided in the original paper's appendix [WAZ24] for further reference.

4.2.2 Self-selective RAG for Code Completion

Central to our framework is the idea of *selective RAG*, where the system decides whether the LM's generation could benefit from retrieved contexts and abstains from retrieval augmentation when it is deemed unnecessary (Figure 4.1).

For this selective decision, two traditional heuristics are relevant: (1) performing a *trial*

retrieval and only augmenting the high-relevance contexts (e.g., Drozdov et al. [DWR22]) or (2) performing a *trial generation* and conducting RAG only when the model's uncertainty is high (e.g., Jiang et al. [JXG23]). For repository-level code completion, these strategies are informative to some extent: in line completion and API completion from RepoEval, both heuristics can maintain the same level of performance with only 50% retrieval budget. However, we find that they fail to generalize well to all tasks and still incur a high latency cost as they need to conduct retrieval to make the decisions (the original paper's appendix [WAZ24]).

Instead, our framework adopts a *self-selective RAG* formulation. After observing X_l and X_r, the LM directly self-triggers cross-file retrieval by generating a special token `<cc>` or abstains from retrieval via an empty token ϕ[1]. This approach is inspired by the explorations in Kadavath et al. [KCA22], which show that an LM can be trained to predict whether it knows the answer or not without retrieval. Beyond this *self-knowledge*, our model also combines the *question's characteristics* (i.e., whether retrieving cross-file information can likely help or not) in its judgment, as we will discuss in the next section. Finally, after the optional retrieval, the LM proceeds with the code completion with X_l, X_r, combined with CC if retrieval is triggered.

Implementation-wise, self-selective RAG's inference is conveniently modeled as an extension to fill-in-the-middle [BJT22], with the entire process executed in a single left-to-right pass (Figure 4.2). One advantage of this design is the *flexibility*. The LM possesses the ability for RAG and fill-in-the-middle, and can seamlessly self-switch between the two when encountering different questions. Users can also easily adjust the ratio between the two through the retrieval threshold. Another advantage is its *efficiency*. The selective decision overhead is only a single forward pass, a significant save compared to making the retrieval decision via trial generation or trial retrieval. When the LM abstains from retrieval, it can directly proceed with generation and the retrieval overhead is completely avoided.

[1]In practice, instead of greedily decoding `<cc>`, we check whether its probability exceeds a certain threshold.

4.2.3 Self-supervised Multi-task Learning

To power self-selective RAG, the LM needs two crucial abilities: accurate self-assessment and robustness to the retrieved context. We design a contrastive data labeling scheme to mine self-supervision from public repositories, followed by fine-tuning with a multi-task objective.

Data construction We leverage large-scale permissively licensed repositories from the Stack [KLA22] and create the fine-tuning data via a three-step procedure:

1. **Sample** target lines Y that are either (1) random code chunks of varied lengths or (2) function bodies.
2. **Retrieve** CC using the current file. We include Y in the query for 50% of the data[2].
3. **Label** whether extending the current file with CC can improve a code LM $\mathcal{M}$'s code completion quality by more than a threshold T, measured by Edit Similarity (ES, definition in Section 4.3.1) against Y.

The fine-tuning instances have the form $(X_l, X_r,\ Y,\ CC,\ \mathit{label})$.

Verbalization Each instance is verbalized into a sequence for fine-tuning. If *label* is false, only X_l and X_r are provided preceding Y. Otherwise, we additionally provide CC after the special token `<cc>`. The two verbalizations correspond to the two branches in Figure 4.2 (b).

Training Objective We introduce two losses, $\mathcal{L}_{eval}$ for self-assessment and $\mathcal{L}_{gen}$ for code generation.

1. $\mathcal{L}_{eval}$: a cross-entropy loss on predicting `<cc>` immediately following `<eof>`.

$$\mathcal{L}_{eval} = -\log p_{\mathcal{M}}(\texttt{<cc>}|X_l, X_r) \tag{4.1}$$

[2]The main goal of the design is to align better with both non-iterative and iterative RAG use cases. During testing, a user may retrieve with both the in-file context and Y', a model's draft prediction, which results in a CC distribution close to that with Y in the query [ZCZ23].

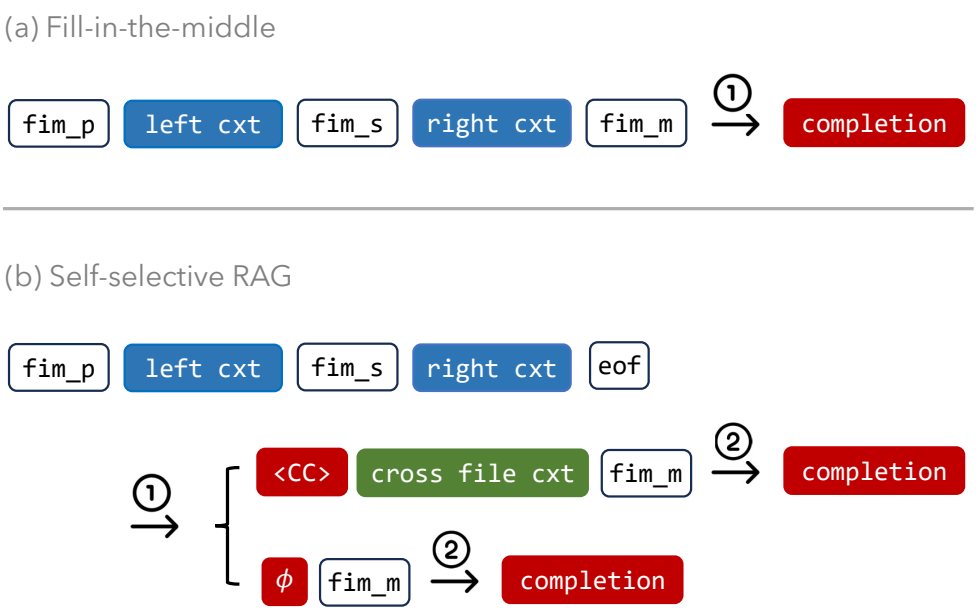


Figure 4.2: A comparison between fill-in-the-middle and self-selective RAG. We mark the end of the current file with a new token `<eof>`, which triggers the LM's self-evaluation. $\rightarrow$ denotes the invocation of the LM. We color current-file context, retrieved contexts, and LM-generated parts in blue, green, and red respectively. `fim_p`, `fim_s`, and `fim_m` refer to the special tokens for fill-in-the-middle: `fim_prefix`, `fim_suffix`, and `fim_middle`. These tokens are already learned during the pre-training.

2. $\mathcal{L}_{gen}$: a cross-entropy loss on the tokens following `<fim_middle>`. Depending on *label*, $\mathcal{L}_{gen}$ represents either code completion with only in-file information or retrieval-augmented code completion.

$$\mathcal{L}_{gen} = \begin{cases} -\log p_{\mathcal{M}}(Y|X_l, X_r, CC), & \text{if } label \\ -\log p_{\mathcal{M}}(Y|X_l, X_r), & \text{otherwise} \end{cases} \tag{4.2}$$

The final training objective is $\lambda\mathcal{L}_{eval} + \mathcal{L}_{gen}$, a weighted combination of the two losses. We do not supervise the model on predicting the other tokens in X_l, X_r, CC, or the special tokens for fill-in-the-middle. Teacher forcing follows normal causal language model training.

4.3 Experimental Setup

4.3.1 Repoformer Implementation Details

Training Data We sample Python repositories from the Stack [KLA22]. Basic filtering are applied to retain 18k repositories that have (1) at least five Python files, (2) at least three imports per file, and (3) at least two local imports per file. These criteria ensure the existence of local dependencies where RAG could be helpful. We use $\mathcal{M}$ = StarCoderBase-1B and $T = 0$ to label 240k chunk and 120k function completion instances. We reserve 500 repositories for validation and use the rest for training.

Training We fine-tune the 1B, 3B, 7B, and 16B variants of StarCoderBase with $\lambda = 1.0$, maximum sequence length 2048, learning rate 2e-5, batch size 512, 50 warmup steps, and a linear learning rate decay. The models are trained for 2 epochs, which approximately takes 8, 12, 20, and 50 hours for the 1B/3B/7B/16B models respectively with 8 Nvidia A100 GPUs (40G memory). Our implementation is based on Jain et al. [JZA23][3]. We will call our models Repoformer-1B/3B/7B/16B. We have also applied the same method to train a multilingual version of Repoformer on a mixture of Python, Java, C#, and Typescript repositories. As we focus on the methodological discussion in the main text, we refer interested readers to Section 4.5.1 for the detailed experiment setup and results.

4.3.2 Evaluation Setup

Evaluation Datasets We evaluate on RepoEval [ZCZ23], which consists of line, API, and function completion tasks created from 32 Python repositories. To investigate the generalization to other languages, we also evaluated the original CrossCodeEval [DWA23], which features line completion instances covering four languages: Python, Java, C#, and Type-

[3] `https://github.com/amazon-science/ContraCLM`



Figure 4.3: The performance gain on RepoEval API completion from retrieved cross-file contexts. Each bucket contains values ranging from label-10 to label+10 except for the central bucket, which corresponds to exactly 0. The retrieved contexts only improve the performance in about 20% of instances. The trend is consistent across all the evaluated LM families and sizes.

Script (Section 4.5.1). Observing that RepoEval has a limited repositrory coverage and that CrossCodeEval has a limited task coverage, we additionally leverage 1500 raw Python repositories from CrossCodeEval to create a new chunk and function completion benchmark, which we call CrossCodeLongEval. We detail the dataset creation process and basic statistics in Section 4.3.3. For the rest of this chapter, we will use CCEval to refer to both CrossCodeEval and CrossCodeLongEval interchangeably, and use the specific language and task (line, chunk, or function completion) to differentiate them.

Evaluation Metrics We evaluate $\hat{Y}$ with both *reference-based* and *execution-based* evaluation. For reference-based evaluation, exact match (EM) and edit similarity (ES) are reported. Following Zhang et al. [ZCZ23], ES is defined as

$$ES(\hat{Y}, Y) = \frac{1 - Lev(\hat{Y}, Y)}{\max(|\hat{Y}|, |Y|)}, \tag{4.3}$$

where Lev is the Levenshtein distance [Lev66]. We report $ES \times 100$ in all the tables following Zhang et al. [ZCZ23] for better readability. For execution-based evaluation, we report the unit test pass rate (UT). $\hat{Y}$ is said to pass the unit tests if replacing Y with $\hat{Y}$ does not cause any unit test to fail. We implement simple post-processing procedures to handle common cases such as excessive lines in model's outputs, which are documented in the original paper's appendix [WAZ24].

Models We experiment on two families of strong code LMs. **CodeGen-Mono** [NPH23] is pretrained sequentially in natural language, multilingual code, and a Python corpus. **StarCoder** and **StarCoderBase** [LAZ23] are trained with fill-in-the-middle ability on a large corpus of multilingual code, GitHub issues, Git commits, and Jupyter notebooks. StarCoder is obtained by training StarCoderBase on an additional Python corpus.

4.3.3 CrossCodeLongEval Construction

One drawback of RepoEval is its limited repository coverage. To verify the performance on diverse repositories, we collect a new evaluation dataset for repository-level code completion.

- **Repository collection.** We first solicited 1744 raw Python repositories from the authors of CrossCodeEval [DWA23]. These repositories were created between 2023-03-05 to 2023-06-15 and collected on 2023-09-01. They have been ensured to not overlap with the Stack [KLA22].
- **Target line sampling.** We avoided using the CrossCodeEval benchmark as the original benchmark explicit removed the instances where StarCoderBase-1B can correctly

	RepoEval			CrossCodeLongEval	
	Line	**API**	**Function**	**Chunk**	**Function**
# repositories	16	16	16	944	1460
# instances	1600	1600	455	5000	5000
$\|X_l\|_{line}$	30.7	30.8	31.1	24.7	31.7
$\|X_l\|_{token}$	796.3	890.7	761.1	661.9	672.1
$\|X_r\|_{line}$	15.1	13.9	16.2	12.9	14.4
$\|X_r\|_{token}$	449.9	430.4	412.4	404.2	371.3
$\|Y\|_{line}$	1.0	2.1	7.8	1.47	9.5
$\|Y\|_{token}$	12.0	25.4	97.8	19.2	111.2

Table 4.1: Descriptive statistics of RepoEval and CrossCodeLongEval. For $|Y|$, $|X_l|$, and $|X_r|$, we report both the number of lines as well as the number of tokens (using the StarCoder tokenizer) in the groundtruth, left context, and the right context.

answer without the retrieved context. To simulate a more natural distribution of code completion, we sample new blanks from the raw repositories. Specifically, we run the chunk- and function-sampling algorithms in the original paper's appendix [WAZ24] to gather chunk completion and function completion instances.

- **Data analysis** In Table 4.1, we present the basic statistics of RepoEval and CrossCodeLongEval.

4.4 Results

4.4.1 Is retrieval always helpful?

As a proof of concept, we first show that on a range of repository-level code completion tasks, the retrieved contexts often fail to improve code LMs' generation quality.

In Table 4.2 and Figure 4.3, we evaluate four code LMs on function completion and API completion from RepoEval. For each model, we report the instance-level performance change from code completion only using X_l and X_r to retrieval-augmented code completion with X_l, X_r, and CC (detailed prompts in the original paper's appendix [WAZ24]).

The results reveal an intriguing pattern: for repository-level code completion, the help

Model	Size	Performance (UT)		UT Change		
		$X_l + X_r$	$X_l + X_r + CC$	$\downarrow$	$=$	$\uparrow$
CodeGen-Mono	16B	23.74	24.18	23	407	25
CodeGen-Mono	2B	30.55	32.51	18	400	37
StarCoder	16B	34.73	42.86	16	386	53
StarCoderBase	1B	22.20	25.71	16	407	32

Table 4.2: The performance change on RepoEval function completion exhibited by four models from retrieved cross-file contexts. For the majority of the instances, RAG does not improve the performance. "↑", "=", "↓" denote the counts for performance increase, no performance change, and performance drop.

from cross retrieval is often sparse. Specifically, retrieval improves LMs' performance on only 20% or fewer instances. For more than 60% of the instances, retrieval augmentation does not affect the performance at all[4]. Finally, another 20% retrievals actually harm the performance, almost as often as the first case. The observed trends are consistent for both API and function completion and hold for both small-sized (1B and 2B) and moderate-to-large (around 16B) code LMs. The generality of this observation is further confirmed by an analysis of Repoformer's training data, where we find that retrieval improves the performance for only fewer than 30% instances (the original paper's appendix [WAZ24]). Together, these findings highlight the suboptimality of the always retrieving and augmenting the cross-file contexts and thus motivate our selective retrieval proposal.

4.4.2 Repoformer achieves strong code completion performance via selective RAG

Next, we evaluate the code completion performance of Repoformer. We compare the following three settings[5]. For the first two baselines, we use the state-of-the-art single-iteration prompting pipeline (Zhang et al. [ZCZ23], detailed in the original paper's ap-

[4]Upon a manual inspection, we find that most of the outputs in this category are also not changed by retrieval at all.

[5]We do not consider iterative retrieval because we find that single-iteration RAG already achieves the majority of the performance gains from multi-iteration RAG.

RepoEval								
Size	**Model**	**RAG policy**	**Line**		**API**		**Function**	
			EM	ES	EM	ES	UT	ES
1B	StarCoderBase	No	43.44	67.77	37.81	66.54	22.20	47.65
		Always	51.19	72.30	43.94	69.17	25.71	55.64
	Repoformer	$\text{Selective}_\text{G}$	51.90	74.50	43.50	71.00	24.00	53.10
		$\text{Selective}_\text{T}$	**54.40**	**76.00**	**46.10**	**72.70**	**28.79**	**57.30**
3B	StarCoderBase	No	49.00	72.12	40.44	69.02	24.84	51.22
		Always	56.69	76.68	47.00	72.62	29.67	57.68
	Repoformer	$\text{Selective}_\text{G}$	56.30	77.60	46.10	73.60	28.57	54.70
		$\text{Selective}_\text{T}$	**59.63**	**79.02**	**49.31**	**74.96**	**32.96**	**60.56**
7B	StarCoderBase	No	51.88	74.03	43.31	70.79	25.49	52.28
		Always	59.44	78.15	49.56	73.65	31.43	58.51
	Repoformer	$\text{Selective}_\text{G}$	56.00	76.63	48.06	75.03	30.77	55.27
		$\text{Selective}_\text{T}$	**59.63**	**78.63**	**50.87**	**76.89**	**35.16**	**60.64**
16B	StarCoder	No	55.25	76.07	44.50	71.00	34.73	53.60
		Always	61.25	79.24	51.12	74.50	42.86	60.96
	Repoformer	$\text{Selective}_\text{G}$	58.13	78.81	48.69	76.23	42.42	58.42
		$\text{Selective}_\text{T}$	**61.75**	**80.34**	**51.88**	**77.93**	**44.18**	**62.58**

Table 4.3: Experiment results on RepoEval and CrossCodeLongEval. The best performance among each model size is boldfaced. We use $\text{Selective}_\text{G}$ and $\text{Selective}_\text{T}$ to denote the greedy selection and the threshold selection strategy for selective retrieval. Repoformer greatly outperforms StarCoderBase of the same size while consuming a smaller retrieval budget. Among the two selective policies, threshold selection enables the best selective RAG performance.

pendix [WAZ24]). We use StarCoder models due to their strong performance among the open-source code LMs.

1. **No Retrieval**. This baseline only provides X_l and X_r to the model in the prompt.
2. **Always Retrieving**. This baseline always augments X_l and X_r with the retrieved CC.
3. **Selective Retrieval**. We provide Repoformer with X_l and X_r in the prompt, optionally augmented with CC based on two selective RAG policies:
   - **Greedy Selection**. Retrieval is performed if `<cc>` is the most likely token following `<eof>`.
   - **Threshold Selection**. If the probability of `<cc>` following `<eof>` is greater than

CrossCodeLongEval					
Size	**Model**	**RAG policy**	**Chunk**		**Function**
			EM	ES	ES
1B	StarCoderBase	No	31.08	60.09	47.49
		Always	37.22	63.73	50.50
	Repoformer	$\text{Selective}_\text{G}$	38.52	68.08	52.09
		$\text{Selective}_\text{T}$	**41.92**	**69.97**	**53.71**
3B	StarCoderBase	No	36.14	64.65	49.88
		Always	42.26	67.74	53.39
	Repoformer	$\text{Selective}_\text{G}$	42.06	70.70	54.47
		$\text{Selective}_\text{T}$	**46.66**	**72.23**	**56.24**
7B	StarCoderBase	No	38.88	66.61	52.45
		Always	44.44	69.53	55.41
	Repoformer	$\text{Selective}_\text{G}$	43.80	72.46	56.14
		$\text{Selective}_\text{T}$	**46.88**	**74.20**	**57.18**
16B	StarCoder	No	42.58	69.40	54.20
		Always	47.90	71.90	58.06
	Repoformer	$\text{Selective}_\text{G}$	45.00	73.36	57.71
		$\text{Selective}_\text{T}$	**49.18**	**75.50**	**58.93**

Table 4.3: Experiment results (continued): CrossCodeLongEval.

a threshold T, retrieval augmentation is performed[6].

The results are summarized in Table 4.3. Compared to no retrieval and always retrieving with StarCoderBase of the same size, Repoformer's selective retrieval strategy exhibits strong performance improvements across all the tasks and both lexical-based and execution-based metrics. Via the threshold selection strategy, Repoformer-3B can outperform StarCoderBase-7B on most of the tasks and metrics except EM for API completion, even outperforming the 5x larger StarCoder in terms of ES for API and chunk completion. Finally, The Repoformer-16B model outperforms the strongest StarCoder baseline by 3%, averaged across all tasks, setting up the new start-of-the-art for repository-level code completion. We also experimentally confirm that the performance improvement from our framework can generalize to three *languages beyond Python* (Section 4.5.1) as well as *dense retrieval* instead

[6]We find that $T = 0.15$ for function completion and $T = 0.2$ for the other tasks generally work well. These two thresholds are always used unless otherwise stated.

of Jaccard similarity (Section 4.5.2). In later sections, we demonstrate that the observed success is due to both the ability to accurately abstain from retrieval and the improved robustness to retrieval.

In terms of code completion accuracy, the threshold selection strategy outperforms the greedy selection strategy on all the tasks. In the next section, we show that the two strategies represent different ways to achieve a good balance between accuracy and inference budget.

4.4.3 Repoformer improves inference efficiency

We illustrate the benefits of REPOFORMER for saving the inference latency in a realistic "online serving" setting.

Latency Model We assume that indexing has already been done for the working repository. Given a code completion request containing the current file (X_l, X_r), the system issues three processes at the same time:

- P_1: make a retrieval decision using REPOFORMER.
- P_2: use a code LM $\mathcal{M}$ to generate $\hat{Y}$ without CC.
- P_3: retrieve CC and generate $\hat{Y}$ with CC using $\mathcal{M}$.

Depending on the result of P_1, the system waits for either P_2 or P_3 and ignores the other process. If $\mathcal{M}$ is REPOFORMER, P_1 can be merged with P_2 by forcing $\mathcal{M}$ to generate a hypothesis without CC after collecting the retrieval decision. We consider three latency terms: (1) T_d, time required for the retrieval decision, (2) T_r, the retrieval latency, and (3) T_g, the generation latency. Then, the latency for P_1, P_2, and P_3 are T_d, T_g, and $T_r + T_g$. When $\mathcal{M}$ is REPOFORMER or a model larger than REPOFORMER, we have $T_d < T_g < T_r + T_g$. Therefore, the latency of the entire system is T_g or $T_r + T_g$ depending on P_1. Using this latency model, we benchmark the latency of various selective retrieval settings on RepoEval with the vllm library [KLZ23] on a single Nvidia A100 GPU (80G).

	RAG Policy	API Completion			Line Completion		
		ES	**%RAG**	**SU**	**ES**	**%RAG**	**SU**
1B	Always	72.02	100%	0%	75.91	100%	0%
	$\text{Selective}_\text{G}$	71.04	18%	69%	74.50	19%	61%
	$\text{Selective}_\text{T}$	72.72	61%	28%	76.00	62%	27%
3B	Always	74.66	100%	0%	78.68	100%	0%
	$\text{Selective}_\text{G}$	73.60	19%	46%	77.60	20%	43%
	$\text{Selective}_\text{T}$	74.96	78%	17%	79.02	74%	16%

Table 4.4: RAG latency of REPOFORMER with two self-selective RAG paradigms. **%RAG** = ratio of instances where RAG is performed. **SU** = Speedup compared to always retrieving (the higher, the better). Compared to the always retrieving baseline, the threshold selection strategy consistently demonstrates gains in both accuracy and latency. The greedy selection strategy shows much larger latency gains with a small performance degradation.

First, we consider $\mathcal{M}$ = REPOFORMER and present the results in Table 4.4. Line and API completion are presented to cover short and moderate target lengths[7]. Both selective strategies significantly improve the latency, with a different trade-off: threshold selection results in *improvements for both accuracy and latency* compared to always retrieving, while using greedy selection results in a larger latency gain with a minor performance degradation (around 1.0 ES). It is worth mentioning that the latency improvement from selective RAG could be further enhanced with a more advanced retrieval setup. For instance, conducting dense retrieval on large repositories often consumes more than 80% of the entire RAG pipeline's latency. Then, a 20% RAG policy could translate into more than 70% speedup. We empirically verify this statement in Section 4.5.2.

Next, we consider using diverse larger LMs as $\mathcal{M}$ in the code completion framework and using $\text{selection}_\text{T}$ with REPOFORMER-1B as a *plug-and-play* selective RAG policy to decide whether retrieval should be performed. We experiment on a diverse set of LMs: StarCoderBase, Code Llama [RGG23][8], CodeGen25 [NHX23], and ChatGPT[9]. As shown in

[7]We omit the function completion results as RepoEval uses very small repositories for function completion for easier unit testing.

[8]We accessed the model through Amazon SageMaker (`https://docs.aws.amazon.com/sagemaker/`).

[9]We use `gpt-3.5-turbo-0613` via the OpenAI API.

Model	RAG Policy	API Completion		Line Completion	
		ES	**SU**	**ES**	**SU**
SCB-7B	Always Retrieving	73.65	0%	78.15	0%
	REPOFORMER-1B	74.10	24%	78.31	25%
SCB-16B	Always Retrieving	74.50	0%	79.24	0%
	REPOFORMER-1B	74.84	24%	79.48	24%
CG25-7B	Always Retrieving	63.07	0%	68.42	0%
	REPOFORMER-1B	63.37	20%	68.86	29%
CL-7B	Always Retrieving	58.75	0%	59.99	0%
	REPOFORMER-1B	58.91	25%	60.47	28%
CL-16B	Always Retrieving	61.08	0%	61.58	0%
	REPOFORMER-1B	62.10	32%	62.45	30%
CHATGPT	Always Retrieving	63.38	0%	61.76	0%
	REPOFORMER-1B	64.01	28%	61.92	18%

Table 4.5: Accuracy and latency of larger code LMs as the generation model and with REPOFORMER-1B as the policy model for selective RAG. SCB = StarCoderBase, CG25 = CodeGen25, CL = Code Llama. **SU** = Speedup compared to Always Retrieving (the higher, the better). Compared to the Always Retrieving baseline, REPOFORMER's selective decisions improve both the accuracy and latency of these larger LMs.

Table 4.5, the selective predictions from REPOFORMER-1B successfully reduce the inference latency with different larger LMs by approximately 25% *while improving their accuracy*. Collectively, the findings indicate that REPOFORMER has acquired robust selective retrieval capabilities that could generalize to diverse types of code LMs.

4.5 Analysis

In this section, we present further analyses and ablation studies on REPOFORMER-1B.

Is Repoformer sensitive to threshold settings? In Figure 4.4, we present the code completion accuracy and latency of REPOFORMER as a function of the threshold. As the threshold increases, the model's code completion performance first increases due to avoiding potentially harmful retrievals. At threshold 0.4, the model still maintains similar performance compared to always retrieving, with latency reduced by 50%. This result demonstrates that REPOFORMER can accommodate various threshold settings and provide

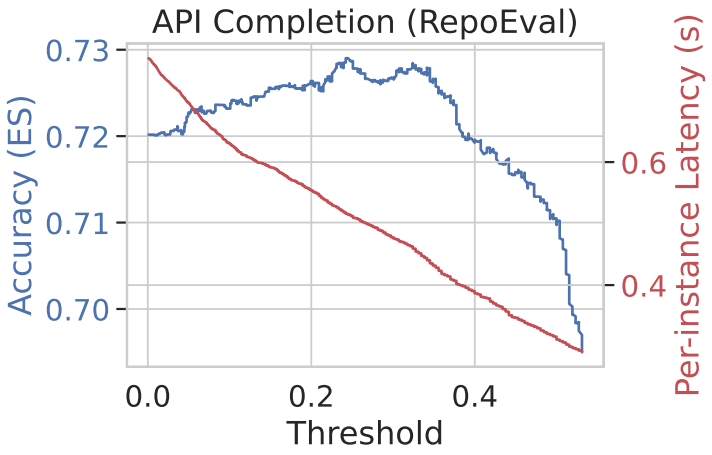


Figure 4.4: The accuracy and latency change with different threshold settings. Selective Retrieval with REPOFORMER achieves better accuracy and better latency than always retrieving. In addition, this behavior is relatively insensitive to the threshold.

a good accuracy-latency trade-off.

Does Repoformer make accurate and calibrated selective retrieval decisions? In Figure 4.5, we evaluate the precision of retrieval abstention decisions made by REPOFORMER's threshold selection strategy. We find that the abstentions are accurate for over 80% instances across all the tasks: when REPOFORMER abstains from retrieval, its code completion prediction either is already correct without retrieval or cannot be improved by retrieval. We also evaluate the calibration of the selective decisions and find REPOFORMER generally making near-calibrated predictions for line and API completion while the calibration is suboptimal for function completion with UT employed as the metric (the original paper's appendix [WAZ24]). We hypothesize that this could be caused by using ES to create the training signal and encourage future work to devise methods for labeling the quality of function completion more effectively.

Is Repoformer robust to retrieval? In Figure 4.6, we show the performance change caused by *CC* on the instances where REPOFORMER requests for retrieval. Compared to STARCODERBASE, REPOFORMER exhibits more and greater performance gains upon ob-

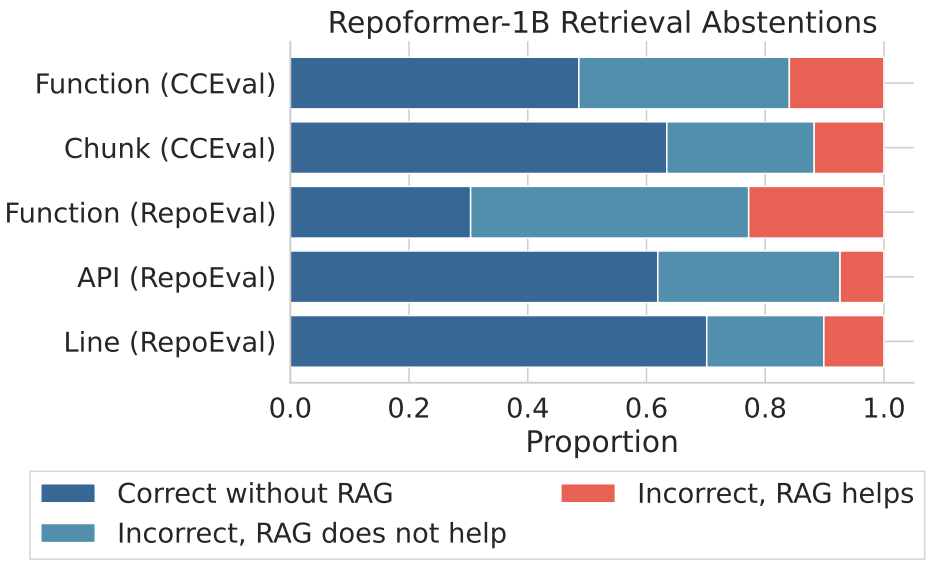


Figure 4.5: An analysis of the instances where REPOFORMER-1B abstains from retrieval. We divide the instances into (1) the model answering correctly without retrieval (dark blue), the model making a mistake that cannot be improved by retrieval (light blue), and the model achieving better performance when retrieval is performed (red). The precision of abstention is over 0.8 on all tasks except for Function (RepoEval), which has a precision of 0.78.

serving CC. The number of performance decreases is also significantly reduced, indicating an improved robustness to the potentially irrelevant retrieval contexts. In Table 4.8 in the appendix, we further study the effect of using *dense retrieval*. Although dense retrieval returns an arguably different context distribution compared to sparse retrieval, REPOFORMER still exhibits strong improvements in both quality and latency.

Ablation Study We study several alternative designs:

- (**A1**) Combining $\mathcal{L}_{eval}$ and $\mathcal{L}_{gen}$ as a single cross-entropy loss. In general, this down-weights $\mathcal{L}_{eval}$.
- (**A2**) Removing the self-evaluation loss $\mathcal{L}_{eval}$.
- (**A3**) Further removing all the CC from A2. This trains only fill-in-the-middle.
- (**A4**) Placing `<cc>` and CC after `<fim_middle>` and marking its end with a new token



Figure 4.6: The performance change on RepoEval from retrieved cross-file context for the instances where REPOFORMER self-selects retrieval. Compared to StarCoderBase, REPOFORMER is better at leveraging *CC* to improve the generation quality.

`<cc_end>`. A4 mainly studies whether it is more beneficial to train the LM to treat *CC* as context fetched during fill-in-middle generation instead of part of the input context.

We fine-tune StarCoderBase-1B with the same setup as REPOFORMER and present the results on CCEval in Table 4.6. Although **A1** has slightly better RAG performance, it fails to make meaningful selective decisions due to $\mathcal{L}_{eval}$ being outweighed by $\mathcal{L}_{gen}$ in long sequences: the probability of `<cc>` is almost always 1. For **A2**, we find it only slightly outperforms REPOFORMER, suggesting learning $\mathcal{L}_{eval}$ does not harm the RAG ability a lot while bringing in the strong selective retrieval ability, which in turn boosts both accuracy and latency. **A3** has the same performance for in-file completion as REPOFORMER, but exhibits

Model	RAG Policy	Chunk Completion			Function Completion		
		T	%RAG	ES	T	%RAG	ES
SC	No	-	0%	60.09	-	0%	47.49
	Always	-	100%	63.73	-	100%	50.50
RF	No	-	0%	66.22	-	0%	49.77
	Selective$_{\text{T}}$	0.20	75%	69.97	0.15	76%	53.71
	Always	-	100%	69.95	-	100%	53.56
A1	No	-	0%	66.14	-	0%	49.25
	Selective$_{\text{T}}$	0.99	100%	70.21	0.99	100%	53.93
	Always	-	100%	70.21	-	100%	53.93
A2	No	-	0%	66.49	-	0%	49.02
	Always	-	100%	70.45	-	100%	53.90
A3	No	-	0%	66.25	-	0%	49.01
	Always	-	100%	68.85	-	100%	52.12
A4	No	-	0%	64.96	-	0%	25.44
	Selective$_{\text{T}}$	0.10	86%	69.35	0.10	83%	26.50
	Always	-	100%	69.19	-	100%	26.35

Table 4.6: Ablation study results. We report the performance on two tasks from the CCEval dataset. **SC** = StarCoderBase-1B. **RF** = REPOFORMER-1B. **T** = threshold for the Selective$_{\text{T}}$ policy. We found T = 0.10 works better for A4 and thus applied it to all the A4 results. **%RAG** = ratio of instances where RAG is performed.

worse RAG performance, indicating the necessity of training with CC. Finally, **A4** achieves reasonable chunk completion performance but performs much worse in function completion. We hypothesize that placing CC within the infilling part is detrimental due to breaking the fill-in-the-middle semantics learned in StarCoder pre-training.

4.5.1 CrossCodeEval and Multilingual Repoformer

This section provides additional results on the 4-language original CrossCodeEval test set [DWA23]. The data creation process of CrossCodeEval explicitly selected the instances where cross-file information is generally required, thus making the contributions from selective retrieval incomplete. On this dataset, we evaluate StarCoder, REPOFORMER-1B/3B/7B trained on Python and REPOFORMER-M trained on multilingual repository-level code completion. Despite the setup difference, we are still able to observe substantial performance gains.

Multilingual Repoformer We experimented with applying the REPOFORMER training scheme to multiple languages. Specifically, we collect public Python, Java, C#, and TypeScript repositories from the Stack [KLA22] that contain at least 20 files and 20,000 lines of code. We do not apply the local import criteria due to implementation difficulties. Then, we follow the algorithm described in the original paper's appendix [WAZ24] to create 90k chunk completion and 30k function completion instances per language. Using this dataset, we fine-tune StarCoderBase following the setup described in Section 4.3.1 (same infrastructure and hyperparameters). We call this model REPOFORMER-M.

Evaluation Results We present the results on CrossCodeEval in Table 4.7 and summarize the observations below:

- **Strong cross-lingual transfer.** REPOFORMER trained on Python data achieves strong performance across multiple languages, including three languages it is not fine-tuned on. The result highlights the generalizability of the learned self-evaluation and robust code completion abilities.
- **Multi-lingual Repoformer.** REPOFORMER-M outperforms the same-sized STARCODERBASE by a large margin. For the 1B, 7B, REPOFORMER-M outperforms REPOFORMER by a small margin. For 3B, the two models give similar performance. This is reasonable as the two models are learned on similar sized training data.

4.5.2 Repoformer's Robustness to the Retriever Choice

In this section, we investigate the performance of REPOFORMER with the cosine similarity of UniXcoder embedding [GLD22] as the retriever instead of Jaccard similarity. As shown in Table 4.8, we are able to observe similar patterns compared to Table 4.4: selective retrieval is able to improve both the accuracy and the latency of the entire RAG system. In addition, as retrieval consumes a larger proportion of latency than when sparse retriever is used, selective

Model	RAG Policy	Python		Java		C#		TypeScript	
		Code ES	ID F1	Code ES	ID F1	Code ES	ID F1	Code ES	ID F1
StarCoderBase-1B	No	68.83	58.18	73.60	63.69	79.30	66.40	67.09	60.15
	Always	71.57	62.42	74.54	65.83	79.04	66.82	67.66	60.60
Repoformer-1B	$\text{Selective}_\text{T}$	71.29	62.81	75.12	67.16	83.08	74.24	69.90	64.07
Repoformer-M-1B	$\text{Selective}_\text{T}$	71.55	62.89	75.92	67.86	84.44	76.00	70.07	64.41
StarCoderBase-3B	No	71.07	61.63	76.10	67.56	81.46	69.95	70.56	64.83
	Always	73.65	65.93	77.52	70.15	81.75	71.26	70.91	65.09
Repoformer-3B	$\text{Selective}_\text{T}$	74.57	66.86	78.40	71.26	85.92	78.62	73.70	68.66
Repoformer-M-3B	$\text{Selective}_\text{T}$	73.80	66.72	77.68	71.01	85.31	77.70	72.51	67.06
StarCoderBase-7B	No	72.47	63.76	77.21	68.97	83.06	72.06	72.34	67.06
	Always	75.02	67.69	77.70	70.57	83.64	74.39	73.01	67.56
Repoformer-7B	$\text{Selective}_\text{T}$	75.34	**68.27**	78.90	72.35	83.80	76.88	73.59	69.10
Repoformer-M-7B	$\text{Selective}_\text{T}$	**75.35**	67.88	**79.11**	**72.82**	**86.53**	**79.77**	**74.60**	**70.01**

Table 4.7: Evaluation results on CrossCodeEval. We report edit similarity for code matching as well as the F1 score for identifier matching. The best scores across all models are boldfaced.

Model	RAG Policy	API Completion			Line Completion		
		ES	%RAG	SU	ES	%RAG	SU
Repoformer-1B	Always	71.69	100%	0%	75.25	100%	0%
	$\text{Selective}_\text{G}$	70.82	18%	71%	73.70	19%	71%
	$\text{Selective}_\text{T}$	72.39	61%	33%	75.65	62%	33%
Repoformer-3B	Always	74.48	100%	0%	78.24	100%	0%
	$\text{Selective}_\text{G}$	73.26	19%	65%	76.74	20%	66%
	$\text{Selective}_\text{T}$	74.69	78%	21%	78.63	74%	31%

Table 4.8: RAG performance of Repoformer with two self-selective RAG paradigms and dense retrieval used instead of Jaccard similarity. **%RAG** = ratio of instances where RAG is performed. **SU** = Speedup compared to always retrieving.

retrieval brings more substantial performance gains, with threshold selection bringing more than 70% speedup.

4.6 Conclusion

In this chapter, we challenge the common assumption of always performing retrieval for RAG-based repository-level code completion. In response, we propose a selective retrieval augmentation framework powered by Repoformer, a code LM that identifies whether cross-file context is necessary, and self-triggers retrieval. Extensive evaluations demonstrate our approach's effectiveness in enhancing accuracy while significantly reducing latency, showcasing its potential in practical coding environments.

CHAPTER 5

Routing Between Parametric and External Knowledge

Retrieval-augmented generation (RAG) equips large language models (LLMs) with external knowledge at inference time, improving performance on tasks that require domain-specific or up-to-date knowledge [KLJ20, LPP20, BMH22, RLD23, SMY24]. However, retrieval can be unnecessary when retrieved passages overlap with the LLM's parametric knowledge, or even harmful when the passages are noisy or irrelevant. These limitations motivate *selective retrieval*, where the system retrieves only when retrieval is likely to be useful [HNB21, MAZ23, XSC24, AWW24, WAZ24].

A limitation of current selective retrieval methods is reducing selective retrieval to a binary decision between consulting a single external source and a no-retrieval fallback where the LLM directly generates the final answer [MAZ23, JBC24, AWW24, WAZ24]. This design provides a coarse characterization of what the model can do without retrieval, as direct answer generation removes an opportunity to explicitly surface relevant parametric knowledge before committing to an answer. Consequently, routers trained by comparing RAG against direct answering, either via answer correctness [WLS23, WAZ24] or likelihood [HNB21, XSC24], can learn preferences inconsistent with the LLM's non-retrieval capability. We observe this in a pilot study, where eliciting and conditioning on verbalized knowledge changes the preferred source for a substantial fraction of instances (Section 5.4.1). Furthermore, the binary setup is inflexible as it cannot learn to leverage multiple knowledge sources, which could be an important limitation in real-world deployments.

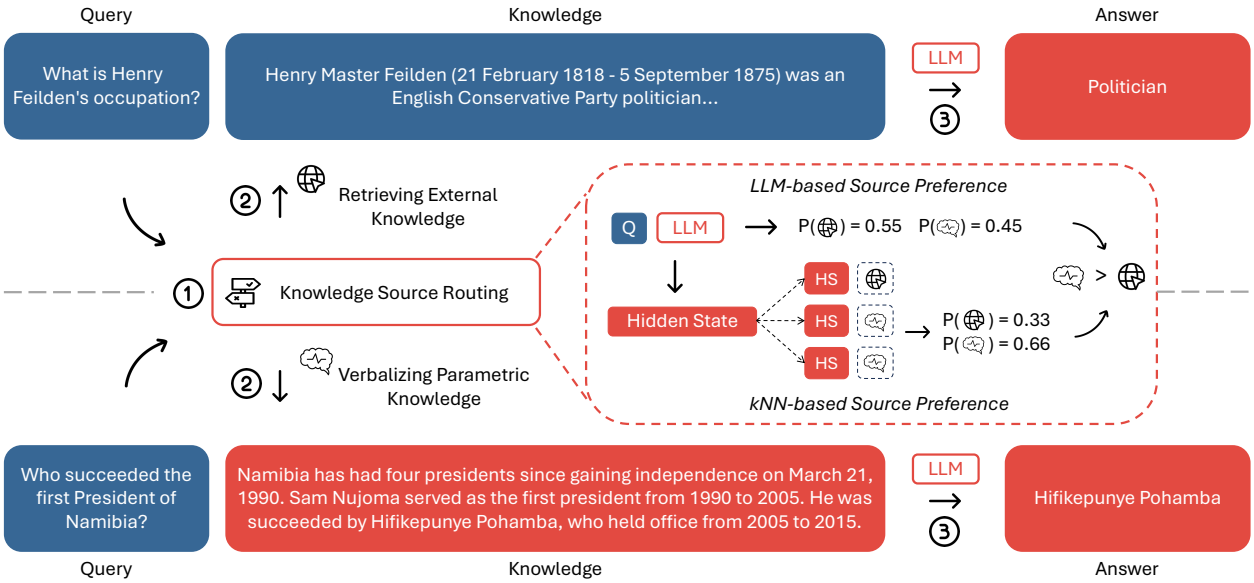


Figure 5.1: **Self-Routing RAG Inference.** (1) A knowledge source is selected based on a query by combining the LLM's likelihood of a special source token with neighbor decisions from an external policy datastore. (2) If an external source is selected, the passages are retrieved and appended as context. If the internal source is selected, the LLM generates a compact verbalized knowledge context. (3) The LLM then produces the final answer conditioned on the query and the selected knowledge. All steps can be executed in a single left-to-right generation pass.

We propose **Self-Routing RAG (SR-RAG)**, which addresses both limitations by casting selective retrieval as a *knowledge source selection* problem and treating the LLM itself as a first-class knowledge source. SR-RAG expands the selective retrieval action space to routing between an internal knowledge source and an expandable set of external sources. As shown in Figure 5.1, at inference time, when the internal source is selected, the LLM first *verbalizes* query-relevant parametric knowledge as a compact context, then answers conditioned on it. When an external source is selected, decoding is paused to retrieve evidence from the selected index, and the model answers conditioned on the retrieved passages.

To make SR-RAG reliable and efficient, we propose three techniques that span data construction, training, and inference. **(1) Verbalization-aware supervision:** given each training question, we collect multiple candidate contexts from each source, including diverse self-verbalized knowledge paragraphs, and assign the preferred source as the one whose con-

texts most increase the likelihood of the gold answer. This produces source selection labels that better reflect the LLM's capability without retrieval than direct-answer baselines. **(2) Multi-task learning:** we jointly train an LLM as a better knowledge source router, a better knowledge source, and a better knowledge reader in RAG. A two-staged multi-task objective is proposed that couples source selection, knowledge verbalization, and answer generation, including a preference alignment training to further boost the knowledge verbalization ability. **(3) kNN-augmented source selection:** purely relying on LLM-based uncertainty to select a source as done in previous work is brittle and agnostic to training-induced LLM ability shifts. We propose to build a policy datastore from light rollouts and combine the model's source selection probabilities with neighbor-defined source distributions in the hidden state space at test time.

We evaluate SR-RAG by fine-tuning Llama-2-7B-Chat [TMS23], Phi-3.5-Mini-Instruct [AJA24], and Qwen2.5-7B-Instruct [YYZ24a] on a mixture of knowledge-intensive tasks and testing on four benchmarks. SR-RAG outperforms a strong standard selective retrieval baseline by 8.5%/2.1%/4.7% while performing 26%/40%/21% fewer retrievals across the three LLMs, respectively (Section 5.4.2). We further show that SR-RAG makes more accurate source selection decisions (Section 5.4.3) and achieves favorable accuracy-latency trade-offs without dataset-specific threshold tuning (Section 5.4.4, Figure 5.4). Ablations confirm that verbalization-aware labeling, preference alignment, and kNN-augmented inference each contribute materially to these gains (Section 5.4.5). Finally, we experiment on routing among three sources (LLM, Wikipedia, and PubMed), where SR-RAG learns to retrieve from specialized corpus when needed while retaining strong performance on general workloads (Section 5.5). Data and code will be publicly released at `https://github.com/xiaowu0162/self-routing-rag`.

5.1 Related Work

Selective Retrieval To enhance the efficiency of RAG systems and reduce harmful or distracting retrievals, several works propose selectively skipping retrieval augmentation [HNB21, MAZ23, XSC24, WAZ24]. A common approach assesses whether retrieval increases the likelihood of generating the correct answer and distills this signal into a supervised decision model [HNB21, SDD23, XSC24]. Similarly, Wang et al. [WLS23] and Wu et al. [WAZ24] compare answers generated with and without retrieval to form supervision based on correctness. Other lines of work examine the question alone to decide whether retrieval is needed [MAZ23, JBC24, AWW24]. To incorporate model confidence, recent methods use uncertainty-based selective retrieval [DPW24, YQP25, MMS25]. In contrast to prior work that treats skipping retrieval as direct answer generation, our work highlights that routing decisions and non-retrieval performance both improve when the LLM is treated as a knowledge source and is trained to verbalize query-relevant parametric knowledge.

Adaptive RAG Inference This work also relates to adaptive RAG, which studies instance-specific inference strategies beyond a fixed retrieve-then-read pipeline. Prior work studies *active retrieval*, where the system re-issues queries when initial retrieval is insufficient [JXG23, STA24]. Another line explores query decomposition and iterative retrieval to handle complex questions by breaking them into sub-queries [SGS23, KKJ23, LPZ24, LAK24]. Given retrieved results, Asai et al. [AWW24] and Yan et al. [YGZ24] critique or revise the retrieved knowledge to improve output quality. Parekh et al. [PPR25] incorporate an initial decision step to adaptively select the most suitable strategy based on the question. While SR-RAG focuses on the targeted decision of which knowledge source to consult, it introduces a complementary form of adaptivity by allowing the model to route among heterogeneous sources, including its own parametric knowledge, within a single generation pass.

LLMs as Knowledge Sources A growing body of work explores using LLMs to generate auxiliary knowledge. Early studies show that LLMs can produce relevant background knowledge in zero-shot settings for commonsense reasoning [SWL20, LLL22]. More broadly, Yu et al. [YIW23] propose a generate-then-read approach, using the LLM as a context generator in place of external retrieval for RAG. LLMs have also been shown to generate effective intermediate reasoning steps for complex questions [WWS22, KGR22]. Building on these insights, our work integrates LLM-generated knowledge into selective retrieval by making it an explicit, routable source, and by training the model to both decide when to rely on it and to produce compact verbalizations that improve end-to-end answering.

5.2 Approach

We first reformulate selective retrieval as knowledge source selection. We then present Self-Routing RAG (SR-RAG), a framework that allows an LLM to self-route between external retrieval and internal knowledge verbalization within a single left-to-right generation pass.

5.2.1 Problem Formulation

Knowledge Source-Aware Inference Given a user query q, a knowledge source s returns relevant information as a text sequence $s(q)$ (possibly empty). An LLM reader M then produces a response conditioned on both the query and the returned knowledge:

$$M(q, s(q)). \tag{5.1}$$

Under this framework, standard RAG is an instantiation with s being a retriever over an external datastore and $s(q)$ being retrieved passages.

Knowledge Source Selection In many practical settings, multiple knowledge sources may be available, denoted by

$$\mathcal{S} = \{\phi, s_1, \ldots, s_N\}, \tag{5.2}$$

where ϕ is a null source that always returns an empty string. A knowledge source selector P chooses the most appropriate source for a given query, i.e., $P(q, \mathcal{S}) \in \mathcal{S}$. Combined with the downstream inference, the full pipeline becomes

$$M\Big(q, P(q, \mathcal{S})(q)\Big). \tag{5.3}$$

Selective retrieval corresponds to the binary case $\mathcal{S} = \{\phi, s\}$, where the selector decides whether to invoke the retriever s or to skip retrieval via ϕ.

5.2.2 Self-Routing RAG: Overview

We propose Self-Routing RAG (SR-RAG), which generalizes selective retrieval by treating the LLM itself as a first-class knowledge source. Concretely, SR-RAG routes between external sources $\{S_{e1}, ..., S_{em}\}$ (e.g., several external corpora) and an internal source S_i (parametric knowledge). If S_i is selected, the model does not directly answer. Instead, it first *verbalizes* query-relevant parametric knowledge into a compact context and then answers conditioned on this context. This makes the non-retrieval path an explicit knowledge consultation step rather than an absence of context.

Following prior work that streamlines adaptive RAG inference via special-token prediction [AWW24, WAZ24], SR-RAG uses three sets of special tokens to implement routing in a single left-to-right generation pass:

1. `<EOQ>`, which marks the end of the query and prompts source selection.[1]
2. A set of source tokens `<s>`, each representing a knowledge source $s \in \mathcal{S}$.[2]

[1]This design aligns with Wu et al. [WAZ24] but differs from Asai et al. [AWW24]. We find the additional `<EOQ>` helps the model allocate probability mass to source tokens without interfering with the query content.

[2]Our main setup uses a simple Wikipedia-based retrieval `<Wiki>` as the only S_e and `<Self>` for S_i. This

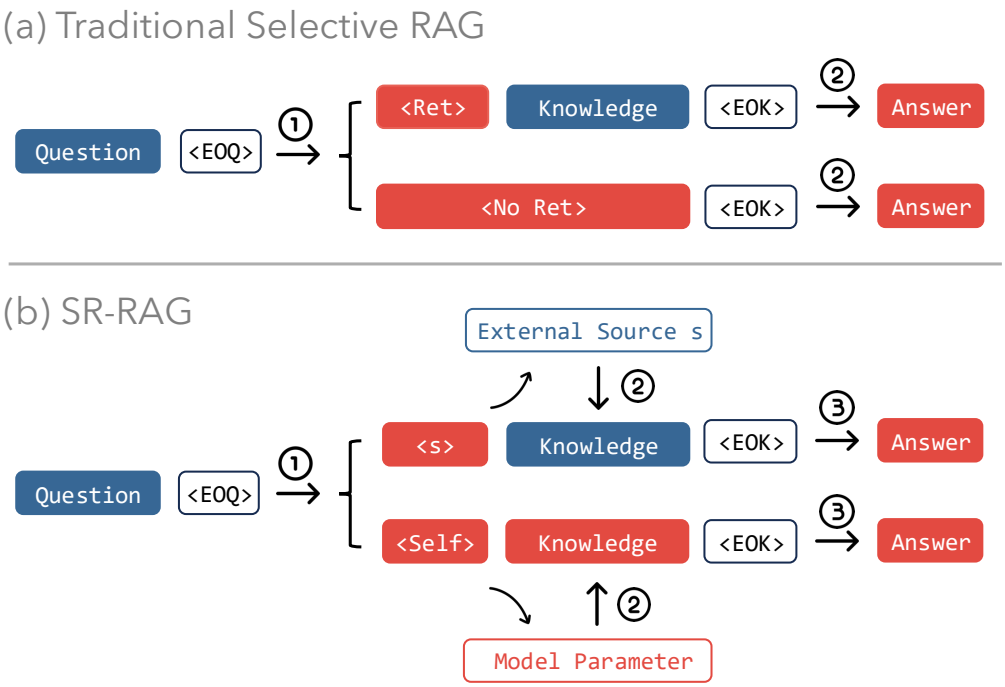


Figure 5.2: **SR-RAG expands selective retrieval into source routing.** Compared to traditional selective retrieval, SR-RAG allows the LLM to route among knowledge sources and to act as a knowledge source itself. Blue denotes external retrieved content and red denotes the LLM and its self-generated tokens.

3. `<EOK>`, which indicates the end of the knowledge context and triggers answer generation.

An illustration of the difference between SR-RAG and standard selective RAG is provided in Figure 5.2.

5.2.3 Self-Routing RAG: Training

We propose a fine-tuning recipe to improve any LLM's ability as the core model in the SR-RAG framework. Inspired by Wu et al. [WAZ24], we propose to mine self-supervision from existing question-answer data (q, a). Our pipeline uses the LLM itself as the internal source

formulation naturally extends to more than two sources by adding additional source tokens. We study a three-source setting in Section 5.5.

S_i along with the external knowledge source, requiring neither human routing labels nor a stronger teacher model. For simplicity, we formulate with $\mathcal{S} = \{S_i, S_e\}$, here. The more general form readily follows and is formulated in Section 5.5.

Data Construction Our central goals are (1) labeling which source is more helpful for a given instance (2) collecting the most helpful knowledge when parametric knowledge is preferred. To do so, our rollout process first collects n candidate contexts from each of the knowledge sources:

- *Parametric knowledge verbalization:* We use GenRead [YIW23] to elicit n diverse verbalizations from the internal source S_i, denoted as $c_{i_1}, \ldots, c_{i_n}$.
- *External knowledge retrieval:* We retrieve n context passages from the external source S_e, denoted as $c_{e_1}, \ldots, c_{e_n}$.

Then, each candidate context c_j is scored by the log-likelihood of generating the gold answer:

$$l_j = \log p_M(a \mid q, c_j). \tag{5.4}$$

We then rank all $2n$ contexts by l_j and select the preferred source $s \in \{S_i, S_e\}$ as the source that appears most frequently among the top-ranked contexts. For convenience, we denote the highest- and lowest-scoring contexts from each source as $c_{i+}, c_{i-}, c_{e+}, c_{e-}$, respectively. The resulting training tuple is $(q, a, s, \{c_j, l_j\})$. The formal data collection algorithm is provided in the original paper's appendix [WGC26].

Objective SR-RAG fine-tunes the LLM to perform (i) source selection, (ii) knowledge verbalization when S_i is selected, and (iii) answer generation conditioned on the selected knowledge. We use a two-stage training procedure.

Stage 1 We perform behavior cloning by jointly optimizing three losses:

1. $\mathcal{L}_{src}$, a cross-entropy loss that trains the model to predict the preferred source token immediately after <EOQ>:

$$\mathcal{L}_{src} = -\log p_M(\texttt{<s>} \mid q). \tag{5.5}$$

2. $\mathcal{L}_{verb}$, a cross-entropy loss over the knowledge tokens when the LLM itself is labeled as the preferred knowledge source:

$$\mathcal{L}_{verb} = \begin{cases} -\log p_M(c_{i+} \mid q), & \text{if } s = S_i, \\ 0, & \text{if } s = S_e. \end{cases} \tag{5.6}$$

3. $\mathcal{L}_{ans}$, a cross-entropy loss on generating the answer conditioned on the preferred knowledge:

$$\mathcal{L}_{ans} = \begin{cases} -\log p_M(a \mid q, c_{i+}), & \text{if } s = S_i, \\ -\log p_M(a \mid q, c_{e+}), & \text{if } s = S_e. \end{cases} \tag{5.7}$$

We combine these objectives as

$$\mathcal{L}_{stage1} = \mathcal{L}_{src} + \mathcal{L}_{verb} + \mathcal{L}_{ans}. \tag{5.8}$$

Stage 2 To further improve the knowledge verbalization quality, we apply Direct Preference Optimization (DPO, Rafailov et al. [RSM23]) using self-generated preference pairs (c_{i+}, c_{i-}):

$$\mathcal{L}_{stage2} = \mathcal{L}_{src} + \mathcal{L}^{DPO}_{verb} + \mathcal{L}_{ans}, \tag{5.9}$$

$$\mathcal{L}^{DPO}_{verb} = \begin{cases} -\log \sigma\Big(\beta \log \frac{p_M(c_{i+}|q)}{p_{ref}(c_{i+}|q)} - \beta \log \frac{p_M(c_{i-}|q)}{p_{ref}(c_{i-}|q)}\Big), & \text{if } s = S_i, \\ 0, & \text{if } s = S_e. \end{cases} \tag{5.10}$$

Here M and ref are initialized from the stage 1 model checkpoint, and only M is updated. Conceptually, stage 2 distills a compute-intensive "system 2" process that searches over multiple verbalizations into a fast "system 1" behavior that produces a single high-quality knowledge verbalization during standard left-to-right decoding.

5.2.4 Self-Routing RAG: Inference

As shown in Figure 5.1 and Figure 5.2, SR-RAG inference proceeds in a single left-to-right generation pass with three steps: source selection, knowledge collection, and answer generation.

Nearest Neighbor-Enhanced Source Selection A common way to choose a source is to threshold the model probability of a source token, such as $p_M(\texttt{<Wiki>} \mid q)$ [AWW24, WAZ24]. In practice, only relying on model's uncertainty can be brittle and does not take into account model ability changes after the fine-tuning. To make source selection more robust and controllable, we augment token likelihoods with a dynamic kNN policy signal derived from additional *rollouts*. Concretely, we evaluate the fine-tuned model on a set of question-answer pairs[3]. For each pair, we compare the likelihood of generating the gold answer under different sources to decide a preferred source label. We then construct a *policy datastore* that maps each query to its preferred source, using the hidden representation at `<EOQ>` as the key. At test time, we retrieve the k nearest neighbors of the query in this representation space and form a neighbor-induced distribution over sources $p_D(\texttt{<s>} \mid q)$ from label counts. Finally, we select the source by applying a threshold on the combined score:

$$p_M(\texttt{<s>} \mid q) \times p_D(\texttt{<s>} \mid q). \tag{5.11}$$

This combination adapts the decision boundary to the fine-tuned model's behavior while preserving the efficiency of a single decoding pass. It also improves SR-RAG's interpretability and controlability as the datastore consists of explicit source assignments, which can be audited, modified, or expanded by practitioners to steer routing behavior.

Knowledge Collection and Answer Generation After selecting a source, SR-RAG gathers knowledge and generates the final answer within the same left-to-right pass. If

[3]Empirically, we reuse the training set so that no additional data or supervision is required.

the internal source S_i is selected, the model uses greedy decoding to verbalize a single compact knowledge context, which serves as a cost-efficient approximation to multi-sample verbalization. If an external source S_e is selected, decoding is paused to retrieve passages from the chosen index, which are appended as the knowledge context. In both cases, the system appends `<EOK>` and the model generates the final answer conditioned on the query and the knowledge.

5.3 Experimental Setup

5.3.1 Implementation Details of SR-RAG

Data Construction The main experiments are performed on two knowledge sources[4]: the 2018 English Wikipedia (`<Wiki>`) as the external knowledge source, and the LLM itself (`<Self>`) as the internal knowledge source. We use the official Wikipedia embeddings released by Karpukhin et al. [KOM20] and retrieve at a granularity of 100-word chunks. GenRead [YIW23] is used to verbalize diverse knowledge contexts. GenRead clusters zero-shot knowledge verbalizations in as in-context demonstrations to verbalize diverse knowledge. We limit the verbalized knowledge chunks to a maximum of 150 tokens. From each knowledge source, we collect $n = 5$ knowledge contexts.

Training We fine-tune on a mixture of six short- and long-form knowledge-intensive datasets: Wizard of Wikipedia [DRS19], Natural Questions [KPR19], FEVER [TVC18], OpenBookQA [MCK18], ARC-Easy [BKK21], and ASQA [SLD22]. This mixture of 53,042 instances is a subset of the RAG instruction tuning data shown effective in Asai et al. [AWW24]. After running the data construction algorithm, 46.9% of the instances are labeled with `<Self>` and the rest are labeled with `<Wiki>` as the preferred knowledge source, averaged across three LLMs. Full training details are provided in the original paper's ap-

[4]We present multi-source extensions of SR-RAG in Section 5.5.

pendix [WGC26].

Inference To construct the policy datastore, we use the representation from a middle layer in the fine-tuned LLM[5], as middle layers are found to be indicative of LLM truthfulness [YSC24]. At test time, the datastore index is cached on GPU and similarity search can be achieved via a single matrix multiplication. We retrieve $k = 30$ nearest supporting examples and construct $p_D(\texttt{<s>}|q)$ from the counts of each knowledge source as the preferred source. Then, we impose a threshold τ on $p_M(\texttt{<Wiki>}|q) \times p_D(\texttt{<Wiki>}|q)$ to decide whether the external knowledge source should be selected.[6]

5.3.2 Evaluation

Datasets and Metrics We evaluate on a diverse set of four knowledge-intensive NLP tasks. **PopQA** [MAZ23] is a free-form long-tail open-domain question answering dataset. Following Asai et al. [AWW24], we use the subset of 1,399 questions that aims to test knowledge of long-tail entities. **TriviaQA** [JCW17] is an established open-domain question answering dataset that features relatively complex and diverse questions. We use the same test split and retrieval setup as in Asai et al. [AWW24]. **PubHealth** [ZLC23] is a fact-checking dataset focusing on the public health domain. Finally, **ARC Challenge** [BKK21] is a multiple-choice question answering dataset featuring grade-school level science questions. Following common practice, we perform lexical post-processing of the model's output and report accuracy for PubHealth and ARC and substring matching for PopQA and TriviaQA.

Baselines We compare SR-RAG with baselines that cover various training and inference strategies. (1) First, using the LLM before fine-tuning, we compare with either always

[5]Layer 15 for Llama-2-7B-Chat and Phi-3.5–Mini-Instruct and layer 11 for Qwen2.5-7B-Instruct. We provide further visualizations and discussions of layer selection in the original paper's appendix [WGC26].

[6]$\tau = 0.1$ for Llama-2-7B-Chat and $\tau = 0.2$ for other models.

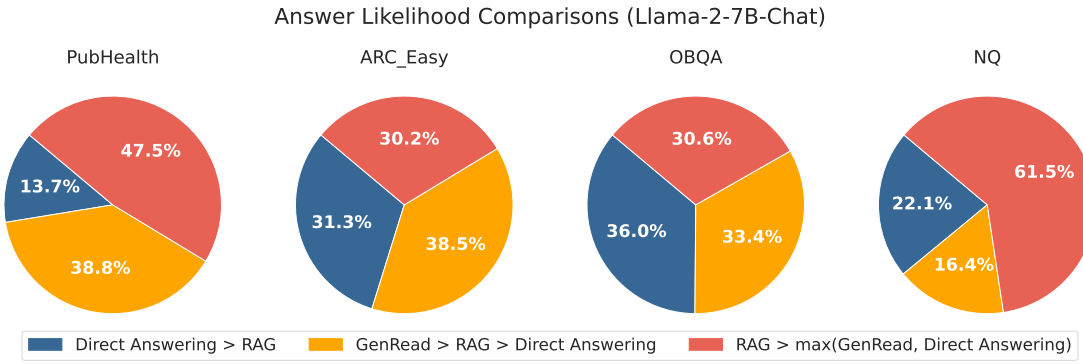


Figure 5.3: **Knowledge verbalization significantly affects the LLM ability boundary.** For a large number of instances (16.4% - 38.8%, orange), GenRead reverses the knowledge source preferences: without considering GenRead, RAG dominates over parametric knowledge.

retrieving or always verbalizing with GenRead. (2) As illustrated in Figure 5.2, our main baseline is the state-of-the-art prior *selective retrieval* pipeline, combining the ideas from He et al. [HNB21], Asai et al. [AWW24], and Wu et al. [WAZ24]. Specifically, the likelihoods of the LLM generating the answer with and without retrieval are used to create the knowledge source selection label. Then, we fine-tune the LLM for knowledge source selection (among S_e and ϕ) and generate the answer with optional retrieval. At inference, we apply a uniform threshold of 0.2 to the likelihood of the retrieval token for selective retrieval. (3) Always retrieving with the fine-tuned LLM after two-staged SR-RAG fine-tuning.

5.4 Results

5.4.1 Knowledge Verbalization Alters Knowledge Source Preference

To motivate SR-RAG, we first show that knowledge verbalization can substantially change when an LLM benefits from retrieval. In a pilot study on Llama-2-7B-Chat using four datasets from our training mixture, we compare the likelihood of generating the gold answer under three conditions: no context (blue), the most helpful GenRead verbalization (c_{i+}, orange), and the most helpful retrieved passage (c_{e+}, red). As shown in Figure 5.3,

Training	Inference	PopQA	TriviaQA	PubHealth	ARC	Average
			Answer accuracy			
			`Llama-2-7B-Chat`			
No Fine-tuning	Always RAG	0.529	0.641	0.457	0.546	0.543
	GenRead	0.247	0.616	0.515	0.605	0.496
Selective RAG	Always RAG	0.567	0.640	0.588	0.588	0.596
	Selective RAG	0.565	0.638	0.589	0.594	0.597
SR-RAG	Always RAG	**0.568**	**0.669**	0.689	0.608	0.634
	SR-RAG	0.566	0.664	**0.730**	**0.630**	**0.648**
			`Phi-3.5-Mini-Instruct`			
No Fine-tuning	Always RAG	0.541	0.594	0.549	0.771	0.614
	GenRead	0.331	0.567	0.442	0.840	0.545
Selective RAG	Always RAG	**0.570**	0.645	0.701	0.813	0.682
	Selective RAG	**0.570**	0.638	0.704	0.815	0.682
SR-RAG	Always RAG	0.567	**0.659**	0.689	0.820	0.684
	SR-RAG	0.566	0.657	**0.705**	**0.854**	**0.696**
			`Qwen2.5-7B-Instruct`			
No Fine-tuning	Always RAG	0.563	**0.667**	0.446	**0.916**	0.648
	GenRead	0.334	0.626	0.676	0.875	0.628
Selective RAG	Always RAG	0.555	0.654	0.600	0.827	0.659
	Selective RAG	0.529	0.648	0.608	0.835	0.655
SR-RAG	Always RAG	**0.573**	0.662	0.596	0.821	0.663
	SR-RAG	0.572	0.659	**0.682**	0.830	**0.686**

Table 5.1: **Main evaluation results on four tasks.** The best results are boldfaced. Across three backbone LLMs, SR-RAG consistently outperforms selective RAG and always retrieving while using a much lower retrieval budget.

GenRead reverses the preferred knowledge source for a large fraction of instances, including 16% of Natural Questions and over 30% on the other datasets. This indicates that selective retrieval methods that omit verbalization can underestimate the LLM's effective capability without retrieval, leading to suboptimal source preference labels and weaker performance when retrieval is skipped. Additional details of the study setup are provided in the original paper's appendix [WGC26].

Retrieval frequency

Training	Inference	PopQA	TriviaQA	PubHealth	ARC	Average
		`Llama-2-7B-Chat`				
No Fine-tuning	Always RAG	100%	100%	100%	100%	100%
	GenRead	0%	0%	0%	0%	0%
Selective RAG	Always RAG	100%	100%	100%	100%	100%
	Selective RAG	98%	100%	100%	65%	86%
SR-RAG	Always RAG	100%	100%	100%	100%	100%
	SR-RAG	96%	89%	40%	29%	64%
		`Phi-3.5-Mini-Instruct`				
No Fine-tuning	Always RAG	100%	100%	100%	100%	100%
	GenRead	0%	0%	0%	0%	0%
Selective RAG	Always RAG	100%	100%	100%	100%	100%
	Selective RAG	100%	95%	91%	83%	92%
SR-RAG	Always RAG	100%	100%	100%	100%	100%
	SR-RAG	98%	92%	24%	5%	55%
		`Qwen2.5-7B-Instruct`				
No Fine-tuning	Always RAG	100%	100%	100%	100%	100%
	GenRead	0%	0%	0%	0%	0%
Selective RAG	Always RAG	100%	100%	100%	100%	100%
	Selective RAG	88%	93%	82%	78%	85%
SR-RAG	Always RAG	100%	100%	100%	100%	100%
	SR-RAG	99%	89%	34%	46%	67%

Table 5.1: Main evaluation results (continued): retrieval frequency (%RAG).

Method	PopQA	TriviaQA	PubHealth	ARC	Average
Accuracy (Verbalization $\geq$ Retrieval)					
Self-RAG	0.957	0.936	0.867	0.908	0.917
SR-RAG w/o. kNN	**0.959**	0.930	0.869	0.888	0.912
SR-RAG	**0.959**	**0.943**	**0.880**	**0.910**	**0.923**
AUROC (Retrieval > Verbalization)					
Self-RAG	0.489	0.503	0.438	**0.557**	0.497
SR-RAG w/o. kNN	0.490	**0.567**	0.564	0.513	0.534
SR-RAG	**0.577**	0.565	**0.606**	0.533	**0.570**

Table 5.2: **Source selection accuracy**. Llama-2-7B-Chat is used. SR-RAG achieves the best averaged performance in both evaluation settings.

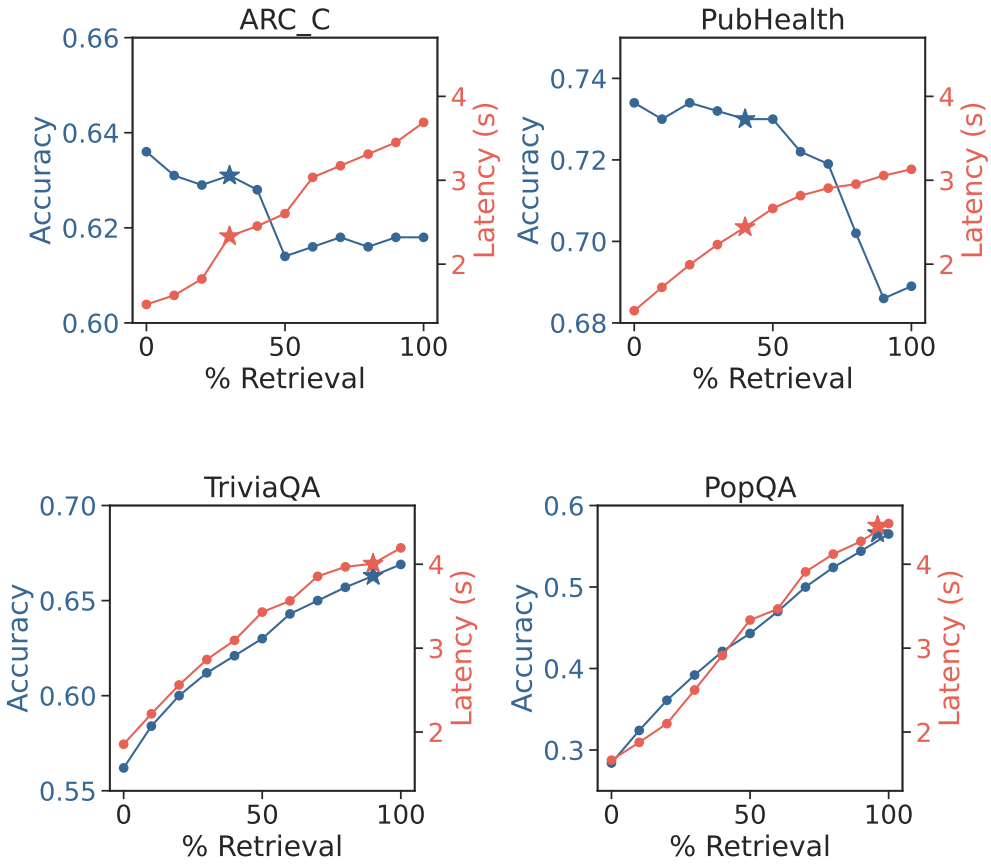


Figure 5.4: **Accuracy and system latency of SR-RAG.** We show the performance of fine-tuned Llama-2-7B-Chat with different verbalization frequencies. SR-RAG's source selection policy (marked with stars) achieves near-optimal accuracy-efficiency trade-off without dataset-specific thresholds.

5.4.2 SR-RAG Improves Accuracy with Reduced Retrieval Frequency

Table 5.1 shows the end-to-end generation performance on three LLMs, demonstrating the advantage of SR-RAG over always retrieving, selective retrieval, and other baselines. Relative to always retrieving with the pretrained model, SR-RAG improves accuracy by 19.3% (Llama-2-7B-Chat), 13.4% (Phi-3.5-Mini-Instruct), and 5.9% (Qwen2.5-7B-Instruct). Compared to the selective retrieval baseline, SR-RAG achieves 8.5%/2.1%/4.7% higher accuracy while triggering 26%/40%/21% fewer retrievals across the three models, respectively. Notably, the selective retrieval baseline saves only 8–15% retrievals and achieves nearly identical

Answer accuracy

Training	Inference	PopQA	TriviaQA	PubHealth	ARC	Average
SR-RAG	SR-RAG	0.566	**0.664**	**0.730**	**0.630**	**0.648**
SR-RAG	SR-RAG w/o. kNN	0.558	0.658	0.694	0.627	0.634
SR-RAG w/o. kv. label	SR-RAG	**0.568**	0.644	0.598	0.629	0.610
SR-RAG w/o. $\mathcal{L}_{verb}^{DPO}$	SR-RAG	0.564	0.645	0.674	0.581	0.616

Retrieval frequency (%RAG)

Training	Inference	PopQA	TriviaQA	PubHealth	ARC	Average
SR-RAG	SR-RAG	96%	89%	40%	29%	64%
SR-RAG	SR-RAG w/o. kNN	94%	77%	72%	56%	75%
SR-RAG w/o. kv. label	SR-RAG	100%	100%	100%	84%	96%
SR-RAG w/o. $\mathcal{L}_{verb}^{DPO}$	SR-RAG	98%	100%	100%	66%	86%

Table 5.3: **Ablation studies on SR-RAG.** Llama-2-7B-Chat is used as the LLM.

accuracy to always retrieving, suggesting that skipping retrieval alone is not sufficient to reliably identify queries that can be handled without external context. In contrast, SR-RAG reduces retrieval by 20–40% while maintaining or improving accuracy, driven by accurate source selection and a stronger non-retrieval path.

Remarkably, without threshold tuning, SR-RAG can *dynamically adapt* its retrieval behavior. For PopQA, which emphasizes long-tail facts, SR-RAG retrieves external knowledge more frequently. For PubHealth and ARC, where the model can often rely on internal knowledge, SR-RAG skips retrieval more and can outperform always retrieving. We find that this flexibility also extends to selecting a specialized external source when multiple are present (Section 5.5).

5.4.3 SR-RAG Makes Accurate Source Selection Decisions

Can SR-RAG choose the appropriate knowledge source? Table 5.2 compares SR-RAG with Self-RAG [AWW24] and an SR-RAG variant without kNN augmentation. We evaluate decision quality under two criteria: (1) abstaining from retrieval when skipping retrieval does not harm performance and (2) retrieving only when retrieval yields strictly better context. Under the first criterion, all methods perform well, and SR-RAG achieves the best overall

accuracy. Under the stricter second criterion, SR-RAG substantially improves AUROC, outperforming Self-RAG by 14.7% on average. Removing kNN augmentation leads to large drops in both accuracy and AUROC, highlighting the benefit of adapting source selection to fine-tuning-induced shifts in model behavior.

5.4.4 System Efficiency

We measure end-to-end latency using Llama-2-7B-Chat under a batched inference setting (details in Section 5.4.6). Figure 5.4 shows the accuracy-latency trade-off as we vary retrieval frequency. As expected, latency improves as fewer instances invoke retrieval, and the optimal retrieval rate varies across datasets. Importantly, SR-RAG's learned policy achieves a near-optimal accuracy-efficiency trade-off across datasets without dataset-specific threshold tuning. These results indicate that SR-RAG improves both accuracy and efficiency in a robust manner.

5.4.5 Ablation Study

We ablate SR-RAG components on Llama-2-7B-Chat in Table 5.3. Disabling kNN-based routing (w/o. kNN) increases retrieval frequency and reduces accuracy, confirming that kNN augmentation improves robustness to ability shifts introduced by fine-tuning. Removing knowledge verbalization during preference labeling (w/o. kv. label) causes the model to over-rely on retrieval and degrades performance, showing that verbalization-aware supervision is important for accurate routing. Finally, ablating stage-2 preference alignment (training stage 2 with the stage-1 objective) reduces verbalization quality and leads to substantially more retrievals, consistent with weaker internal knowledge generation. Together, these results indicate that all three components contribute materially to SR-RAG's overall gains. We provide additional analyses in the original paper's appendix [WGC26] including the choice for source labeling heuristics, the size of the policy datastore, and other hyperparameters.

5.4.6 Latency Formulation

To evaluate the inference efficiency of SR-RAG, we measure the latency in a realistic batched inference setup, where the system handles a batch of $B = 10$ queries and returns the results for all queries. For the latency experiments presented in the paper, we decompose the system latency as follows:

- **Source Selection Time** (T_d): The time for the knowledge source selector to determine whether to retrieve from external sources or rely on parametric knowledge.
- **Retrieval Latency** (T_{rs}): The time for fetching external knowledge from the database if the model chooses to retrieval from an external knowledge source s. In our batched setting, we calculate T_{rs} by performing a batched retrieval for all the B instances that require retrieval and report the per-item latency.
- **Verbalization Latency** (T_v): The time for the LLM to verbalize parametric knowledge if the LLM itself is selected as the knowledge source.
- **Generation Latency** (T_g): The time for the LLM to generate the response, conditioned on either retrieved or verbalized knowledge.

Thus, the total per-item latency T_{total} is given by:

$$T_{\text{total}} = \begin{cases} T_d + T_v + T_g, & \text{if verbalize,} \\ T_d + T_{rs} + T_g, & \text{if retrieve from } s. \end{cases}$$

We have the following remarks:

- This formulation assumes that both the retrieval index and the source selection datastore are pre-constructed. This assumption is reasonable as these indices are only constructed once and then constantly reused.

- We choose the batched setup due to the complexity of the retrieval system. For instance, in our implementation of Wikipedia search, it takes around five to ten seconds per instance to encode the query and retrieve the most relevant context chunks. The batched setting amortizes the latency of single retrieval.

- As the nearest-neighbor search only involves one matrix product for similarity calculation and one top-k operation, T_d is generally very small. In fact, we find T_d (0.01s) $<< T_g$ (0.1s) $< T_v$ (1s). On the other hand, T_{rs} is the major bottleneck of the pipeline. As a result, in an online setting, the system's efficiency gain directly converges to the percentage of retrievals it is able to avoid.

5.5 SR-RAG: Multi-Source Extensions

Beyond routing between the LLM itself and an external knowledge source, SR-RAG's formulation is naturally compatible with routing between multiple external knowledge sources as well. In this section, we provide further experiments on routing between three knowledge sources.

5.5.1 Experiment: Routing between General and Specialized Sources

We perform experiments on the PubMed corpus [Nat], which contains professional biomedical literature. Compared to Wikipedia and the LLM itself, PubMed has the unique advantage of featuring long-tail specialized biomedical knowledge. Meanwhile, it falls short at providing knowledge for many general factuality questions, posing a challenge to accurately routing the knowledge source.

Based on this new setup, we formulate SR-RAG with three knowledge source tokens: `<Self>`, `<Wiki>`, and `<Pubmed>`. We leverage the MedRAG toolkit [XJL24] to implement retrieval on PubMed. The training and inference strategies of the two-source SR-RAG recipe

naturally extends to this formulation and are followed in our experiments here. For training and evaluation, we additionally use PubMedQA [JDL19], a widely used biomedical question answering dataset. PubMedQA features short questions extracted from PubMed abstracts and provides associated ground truth yes/no/maybe answers. We sample 10k artificially generated documents for training (combined with the data mixture used in the main text) and 500 examples from the expert-labeled subset for evaluation. The same set of LLMs from three families (Llama, Qwen, Phi) are trained and evaluated.

In Table 5.4, we present the three-way routing results of SR-RAG and compare with always-RAG as well as GenRead with the model before fine-tuning. Our main findings are as follows:

- **Adding PubMed as an additional knowledge source does not harm its effectiveness.** The results on datasets other than PubMedQA align with the performance of two-source SR-RAG reported in the main text.
- **SR-RAG training improves the model's ability to verbalize its knowledge** on PubMedQA as well while being much more efficient than GenRead, as shown by the better performance of SR-RAG + Always Verbalizing compared to GenRead.
- **SR-RAG intelligently routes the best knowledge source for different workloads.** For PubMedQA, PubHealth, and ARC, where PubMed can outperform Wiki by a large margin, SR-RAG frequently self-selects PubMed as the knowledge source. On the other hand, on general factuality benchmarks such as PopQA and TriviaQA, SR-RAG selects Wikipedia more frequently. More importantly, SR-RAG achieves this behavior end-to-end without the need for any manual interventions.
- **On PubMedQA, the end-to-end SR-RAG achieves the best performance compared to the baselines.** The trend is consistent across three different LLMs.

Overall, these experiments confirm the effectiveness of extending SR-RAG to three sources and suggests a promising outlook for further extension.

5.6 Conclusion

We present SR-RAG, a novel retrieval-augmented generation (RAG) framework that tightly integrates selective retrieval with knowledge verbalization. By reformulating selective retrieval as a knowledge source selection problem, SR-RAG enables the LLM to not only choose between external and internal knowledge sources but also to serve as a knowledge source itself. During inference, SR-RAG leverages internal hidden states and a nearest-neighbor policy to make accurate, adaptive source selection decisions. Extensive experiments show that SR-RAG significantly improves answer accuracy while reducing retrieval frequency and latency, offering a scalable and reliable path forward for more efficient, knowledge-aware RAG systems.

Training	Inference	PubMed QA	PopQA	TriviaQA	PubHealth	ARC	Average
			`Llama-2-7B-Chat`				
No Fine-tuning	GenRead	0.544	0.247	0.616	0.515	0.605	0.505
	Always RAG, Wiki	0.636	0.529	0.641	0.457	0.546	0.562
	Always RAG, PubMed	0.656	0.157	0.470	0.612	0.537	0.486
SR-RAG (3-source)	Always RAG, Wiki	0.342	**0.587**	0.639	0.668	0.639	0.572
	Always RAG, PubMed	0.672	0.325	0.539	0.713	0.645	0.642
	Always Verbalizing	0.388	0.305	0.572	**0.724**	0.631	0.579
	SR-RAG	**0.680**	0.583	**0.647**	**0.724**	**0.655**	**0.677**
	(% `<Wiki>`)	6%	95%	88%	17%	4%	29%
	(% `<Pubmed>`)	91%	4%	7%	70%	86%	64%
	(% `<Self>`)	3%	1%	5%	13%	10%	8%
			`Phi-3.5-Mini-Instruct`				
No Fine-tuning	GenRead	0.376	0.331	0.567	0.442	**0.840**	0.511
	Always RAG, Wiki	0.460	0.541	0.594	0.549	0.771	0.583
	Always RAG, PubMed	0.676	0.143	0.356	0.447	0.726	0.470
SR-RAG (3-source)	Always RAG, Wiki	0.486	**0.580**	0.634	0.676	0.805	0.636
	Always RAG, PubMed	0.738	0.302	0.494	0.644	0.813	0.598
	Always Verbalizing	0.498	0.327	0.510	0.704	0.819	0.572
	SR-RAG	**0.736**	0.560	**0.639**	**0.716**	0.824	**0.695**
	(% `<Wiki>`)	1%	95%	87%	54%	0%	47%
	(% `<Pubmed>`)	97%	3%	10%	18%	27%	31%
	(% `<Self>`)	2%	2%	3%	28%	73%	22%
			`Qwen2.5-7B-Instruct`				
No Fine-tuning	GenRead	0.418	0.334	0.626	0.676	0.875	0.586
	Always RAG, Wiki	0.612	0.563	**0.667**	0.446	0.916	0.641
	Always RAG, PubMed	0.698	0.129	0.469	0.401	0.821	0.504
SR-RAG (3-source)	Always RAG, Wiki	0.358	**0.576**	0.639	0.637	0.818	0.606
	Always RAG, PubMed	0.726	0.307	0.503	0.676	0.824	0.607
	Always Verbalizing	0.522	0.303	0.550	**0.695**	0.828	0.580
	SR-RAG	**0.730**	0.572	0.651	0.681	**0.834**	**0.694**
	(% `<Wiki>`)	15%	96%	86%	20%	0%	43%
	(% `<Pubmed>`)	79%	1%	13%	48%	39%	36%
	(% `<Self>`)	6%	3%	1%	32%	61%	21%

Table 5.4: **Three-source results.** We route between retrieving from **Wikipedia**, retrieving from **PubMed**, or **self-verbalizing** knowledge. For each model we report task accuracy and the selection frequencies of the individual sources `wiki/PubMed/self` under SR-RAG's LLM+kNN knowledge source routing scheme.

Part III

Framing, Discovering, and Evolving Reusable Knowledge

CHAPTER 6

Discovering Domain-Specific Policy Knowledge

Content moderation modules are core layers in modern products for managing unsafe or low-quality inputs. These systems are guided by domain-specific policies that define allowed and disallowed content and support consistent enforcement across labeling, training, and deployment [MZA23, VD20, YZ21, ZKR20]. However, drafting and maintaining high-quality policies remains costly. It requires domain expertise, repeated iteration, and frequent updates as products evolve and new edge cases appear. While studies have explored automatic pipelines to improve the effectiveness or reduce the cost of applying the policies [MZA23, GJW24], a fully human-written policy is still a prerequisite. In this chapter, we challenge this assumption by asking:

Can we leverage LLMs to assist in drafting the policies themselves?

To begin with, we frame the task **open-domain policy construction**. The input is a concise domain specification and access to a search engine. The output is a structured policy document. Success is measured by downstream utility, such as the accuracy of a fixed moderation model when the policy is provided in-context.

We then propose **Deep Policy Research** (DPR). DPR is a minimal agent that uses only web search as an external tool and a lightweight scaffolding scheme for rule writing. Starting from a one-sentence domain definition, DPR iteratively identifies missing coverage, issues targeted queries, distills retrieved sources into rule predicates, and consolidates them into an indexed policy document (Figure 6.1). The indexing stage organizes rules into coherent sections, improving readability and helping reader models consume long policies.

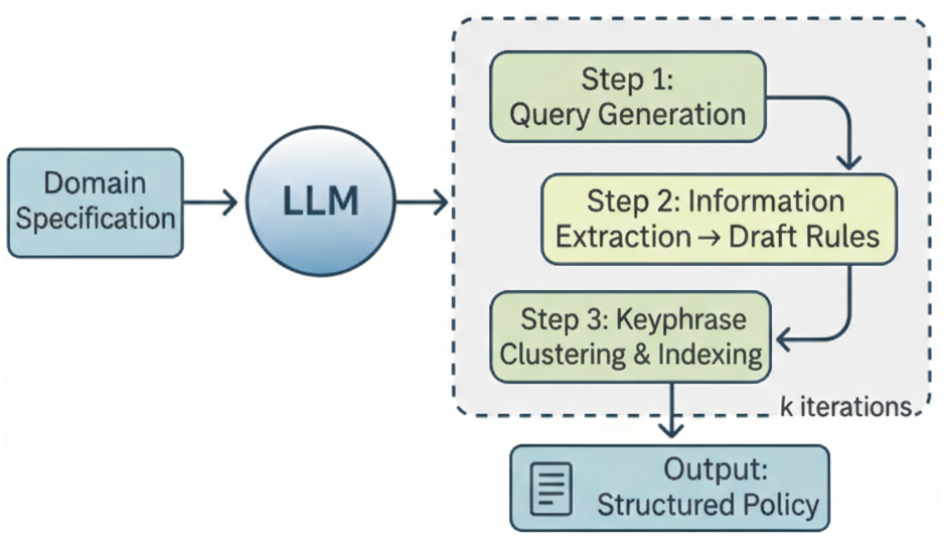


Figure 6.1: An illustration of Deep Policy Research. Based on a domain specification, an LLM iteratively interacts with a search engine, extracts policy rules, and indexes the rules through keyphrase-based clustering.

We evaluate DPR in two content moderation settings. On the OpenAI undesired content benchmark [MZA23] across five domains, DPR improves moderation F_1 over both definition-only prompting and few-shot in-context examples for two compact reader LLMs. Averaged across domains, DPR increases F_1 from 0.752 to 0.792 on Llama 3.1 8B and from 0.810 to 0.831 on Qwen2.5 7B, with the largest gains on more subjective categories such as Violence, Harassment, and Self-Harm (Section 6.3.2). Under the same seed specification and evaluation protocol, DPR also outperforms a general-purpose deep research system, improving average F_1 from 0.776 to 0.792 on Llama 3.1 8B and from 0.800 to 0.831 on Qwen2.5 7B (Section 6.3.2). We further evaluate DPR on an in-house multimodal advertisement moderation benchmark. Replacing an expert-written domain section with a DPR-generated section recovers much of the human policy benefit in several domains and substantially improves over removing the section or using only the one-sentence specification (Section 6.4.2).

In summary, we introduce open-domain policy construction and evaluate it by downstream utility. We propose Deep Policy Research, a minimal agent that uses web search and lightweight scaffolding to synthesize and index domain-specific policies. Across text-only

and multimodal moderation settings, DPR-generated policies improve downstream moderation and are competitive with expert-written policy sections in several domains, while outperforming a general-purpose deep research baseline under the same protocol. DPR also provides a reproducible environment for future work on policy drafting agents, including reader-model-specific policy presentation and generating illustrative examples alongside rule predicates to clarify decision boundaries.

6.1 Related Work

Policy Use in LLM Systems Recent alignment strategies explicitly incorporate human-written safety policies into the training or reasoning process of large language models. Deliberative Alignment fine-tunes LLMs to reason based on an entire written policy before responding, yielding safer outputs that strictly adhere to guidelines (e.g. reduced jailbreaks and fewer unjustified refusals) [GJW24]. Anthropic's Constitutional AI similarly forgoes direct human feedback in favor of a fixed set of normative principles that the model internalizes as a "constitution," using them to self-criticize and refine its answers [BKK22]. Other works integrate policies as part of the reward or decision mechanism: for instance, OpenAI's GPT-4 alignment process included a rule-based reward model that penalized policy violations during RLHF [MHH24], and recent safety reasoning frameworks train models to follow chain-of-thought traces grounded in explicit policy rules [MLZ25].

Policy Writing The direction of automatically creating or refining safety policies has been less explored. OpenAI has demonstrated that GPT-4 can assist policy designers by identifying ambiguities and edge cases in draft guidelines: the model labels content according to a given policy, explains any discrepancies with human judgments, and suggests clarifications, thereby accelerating the policy refinement loop [Ope23b]. We build upon this intuition and take a step forward, automatically generating policy end-to-end from

human-curated domain specifications.

6.2 Approach

6.2.1 Problem Formulation

We study **open-domain policy construction**. The input is a domain specification s that describes scope and intent for a single moderation domain, and a search engine $\mathcal{G}$. The output is a policy document P consisting of a set of textual rules organized into sections. P can be free-formed or following a hierarchical structure. We evaluate P by **downstream utility** in a fixed content moderation setup, where a reader LLM receives P in-context and performs safe vs. unsafe binary classifications. A policy construction system succeeds if its output policy improves downstream moderation performance under the same reader model and evaluation protocol.

6.2.2 Deep Policy Research

We present **Deep Policy Research** (DPR), a minimal research agent that constructs a policy by iteratively searching the web and distilling sources into structured rules. DPR uses an LLM $\mathcal{M}$ as the research model and web search $\mathcal{G}$ as the only external tool. DPR runs for k iterations and maintains two artifacts at iteration i: a policy draft P_i and an index I_i that organizes rules into sections. DPR initializes $P_0 \equiv s$ and $I_0 \equiv s$.

Concretely, at each iteration $i \in \{1, \ldots, k\}$, DPR performs three steps.

Step 1: Query generation. DPR first analyzes the current policy organization I_{i-1} and proposes a set of research queries Q_i to expand coverage or refine ambiguous parts of the policy. Queries are written to target definitional boundaries, common edge cases, high-risk subtypes, and enforcement cues. For each query $q \in Q_i$, DPR retrieves the top m search

results using $\mathcal{G}$ and collects page titles, snippets, and URLs as evidence for rule extraction.

Step 2: Rule extraction and consolidation. Given the retrieved evidence, DPR prompts $\mathcal{M}$ to extract candidate rules in a consistent schema. Each rule is written as a short predicate-style statement with a clear decision boundary and optional qualifiers. Concretely, we ask $\mathcal{M}$ to produce rules that specify a condition and a moderation decision, and to include brief scope qualifiers when needed. DPR then runs a self-critique pass to improve precision and reduce noise. In this pass, $\mathcal{M}$ removes irrelevant or overly generic rules, merges redundant rules that express the same decision boundary, and resolves conflicts by preferring rules that are supported by multiple sources or by higher-quality sources. The output is a consolidated rule set R_i.

Step 3: Indexing. DPR merges the new rules into the policy draft, $P_i \leftarrow P_{i-1} \cup R_i$. It then organizes the full rule set into sections to form an indexed policy document I_i. We use keyphrase-based clustering to build this index. DPR asks $\mathcal{M}$ to extract keyphrases for each rule, clusters keyphrases into n groups using k-means, and asks $\mathcal{M}$ to name each cluster and write a short section summary that captures the shared theme. Finally, DPR merges clusters with overlapping semantics to produce a compact, readable index. The resulting I_i is used both as a human-readable policy document and as a coverage signal for the next query generation step.

After k iterations, DPR outputs the final indexed policy $P \equiv I_k$. Figure 6.1 illustrates the overall loop. Our goal is not to engineer complex agent architectures or elaborate scaffolding strategies. Instead, we aim to isolate a minimal, reproducible research loop that uses a single external tool and lightweight human scaffolding, and to show that this simple design can already produce useful, structured policies with measurable downstream utility. For reproduction, we list the domain specifications and prompts in the original paper's appendix [WLJ26].

	Sexual	Hate	Violence	Harassment	Self-Harm	Average
`Llama 3.1 8B Instruct`						
Seed Information	0.916	0.757	0.658	0.640	0.788	0.752
In-Context Learning	0.923	0.728	0.582	0.610	0.691	0.707
OAI DR	0.828	0.800	0.738	0.683	0.829	0.776
DPR	0.910	0.813	0.717	0.683	0.835	**0.792**
`Qwen2.5 7B Instruct`						
Seed Information	0.939	0.817	0.782	0.670	0.842	0.810
In-Context Learning	0.927	0.803	0.645	0.551	0.779	0.741
OAI DR	0.919	0.765	0.763	0.781	0.773	0.800
DPR	0.949	0.811	0.784	0.752	0.860	**0.831**

Table 6.1: Evaluation results on OpenAI Content Moderation. DPR improves content moderation F1, outperforming both only using seed information or performing in-context learning with human-written examples.

6.3 Results: OpenAI Content Moderation

6.3.1 Experimental Setup

We evaluate DPR on the OpenAI undesired content benchmark introduced in Markov et al. [MZA23]. Following prior work, we consider five major moderation domains and report binary classification F_1 computed per domain and averaged across domains. In all settings, we keep the downstream reader LLM fixed and vary only the policy provided in-context, so differences reflect the utility of the constructed policy rather than changes in the classifier or model weights.

Evaluation We select the five major domains (sexual, hate, violence, harassment, and self-harm) and use their official definition as the input for DPR[1], as presented in the original paper's appendix [WLJ26]. For each domain, we reserve five examples for validation and use the rest for testing. We perform evaluation separately for each domain using the

[1]Accessed at `https://github.com/openai/moderation-api-release`.

in-domain undesired test examples as well as the safe test examples. To define the utility metric for the system-generated policy, we focus the evaluation scope on long-context evaluation with an off-the-shelf LLM. In other words, we directly provide the entire policy in the context of the LLM and prompt for the binary safe/unsafe judgment.

DPR Implementation Details We use the domain definitions provided in the original paper's appendix [WLJ26] as the policy specification s. Unless otherwise stated, we use Llama 3.3 70B Instruct [DJP24] as $\mathcal{M}$ and Google Search as $\mathcal{G}$. We run research for $k = 3$ iterations and use $n = 20$ for clustering.

Baselines We compare DPR with three baselines:

- **Seed Information**: judging with only the seed information s_i for each domain.
- **In-Context Learning**: we randomly sample three unsafe examples and three safe examples as the in-context demonstrations.
- **OAI DR**: we manually run the OpenAI Deep Research Agent through the WebUI. The seed information is provided and we use GPT-5.1 as the research model.

6.3.2 Content Moderation Accuracy

Table 6.1 reports results across five domains and two reader LLMs. DPR consistently improves over both Seed Information and In-Context Learning for every domain and reader model. Averaged across domains, DPR increases F_1 from 0.752 to 0.792 on Llama 3.1 8B and from 0.810 to 0.831 on Qwen2.5 7B. Gains are most pronounced in more subjective categories such as Violence, Harassment, and Self-Harm, where definition-only prompts leave substantial ambiguity and few-shot demonstrations are brittle. Notably, DPR does not introduce regressions on well-specified categories. For Sexual, differences are within 0.01 F_1 for both readers, indicating that web-grounded refinement can add coverage without injecting noise.

Under the same seed specification and evaluation protocol, DPR also outperforms the

general-purpose deep research baseline. On Llama 3.1 8B, the average F_1 improves from 0.776 with OAI DR to 0.792 with DPR. On Qwen2.5 7B, the average improves from 0.800 to 0.831. This suggests that task-specific structure matters for policy drafting: a simple loop that enforces rule extraction, consolidation, and indexed organization can yield more usable policies than generic web research when the end goal is downstream moderation.

Overall, these results support two conclusions. First, open-domain policy construction can translate into measurable moderation gains even when the downstream model is held constant. Second, a minimal agent with a single external tool can be effective when paired with lightweight human scaffolding that constrains outputs into actionable rule predicates and an indexed document. We provide additional analyses on the research model and indexing design in Section 6.5.

6.4 Results: In-house Multimodal Advertisement Moderation

6.4.1 Experimental Setup

We further evaluate DPR in a real-world multimodal moderation setting using an in-house advertisement benchmark. Each example consists of ad text and a thumbnail image, and the task is to predict whether the creative complies with an in-house safety policy. We report binary classification F_1 and evaluate with a fixed vision-language reader model that receives the policy in-context. We consider two inference settings, single-sample decoding (S.S.) and majority voting (M.V.) over multiple samples.

Evaluation We select four domains and provide a one-sentence summary as the input for DPR, as presented in the original paper's appendix [WLJ26]. We perform evaluation separately for each domain using the in-domain safe and unsafe examples. Similar to the previous experiment, we provide the entire policy in the context of an off-the-shelf large vision

and language model[2] and prompt for the binary safe/unsafe judgment. We experiment in two inference settings: *single sample (S.S)* and *majority vote (M.V.)* with 10 responses sampled with temperature 0.8.

DPR Implementation Details We use Llama 3.3 70B Instruct as the research LLM. We keep all the hyperparameter settings consistent with the previous section and the only difference is the input domain specification.

This benchmark includes a comprehensive human-written policy document, where each section corresponds to a policy domain. Our evaluation therefore focuses on **substituting** a single domain section while keeping the remainder of the policy fixed. We compare five configurations:

1. **No Policy** removes the target domain section entirely while keeping the other sections.
2. **Human Policy** uses the full original expert-written policy document.
3. **Seed Information** replaces the section with the one-sentence domain specification.
4. **DPR** replaces the section with a DPR-generated policy built from the same one-sentence specification and web search.
5. **DPR + Summary** further compresses the DPR index into a shorter rule set to reduce prompt length.

6.4.2 Content Moderation Accuracy

Table 6.2 reports results for four domains under both inference settings. Removing the domain section or replacing it with only the one-sentence specification degrades performance in three of four domains, showing that the domain-specific section carries substantial utility beyond the rest of the policy. In contrast, substituting the section with DPR recovers much of this benefit. Under single-sample inference, DPR improves the average F_1 from 0.68 with

[2]We used OpenAI `gpt-4o-2024-08-06`.

	Misrepresentative		Finance Claims		Exploitative		Offensive	
	S.S.	**M.V.**	**S.S.**	**M.V.**	**S.S.**	**M.V.**	**S.S.**	**M.V.**
No Policy	0.714	0.727	0.500	0.509	0.793	0.782	0.714	0.786
Seed Information	0.701	0.701	0.553	0.544	0.839	**0.885**	0.679	0.786
Human Policy	**0.740**	**0.779**	**0.833**	**0.877**	**0.920**	0.908	**0.893**	**0.964**
DPR	0.727	0.740	0.597	0.597	0.908	**0.920**	0.786	0.821
DPR + Summary	0.701	**0.779**	0.588	0.614	0.874	0.908	0.821	0.821

Table 6.2: Evaluation results on in-house multimodal moderation data. DPR improves content moderation F1, outperforming both no policy and only using seed information. DPR also performs on par with human-written policy in the Mispresentative and Exploitative domain. We provide further human evaluations in the original paper's appendix [WLJ26].

No Policy and 0.69 with Seed Information to 0.75. Gains are largest for visually nuanced categories such as Exploitative and Offensive, where the web-sourced rules often capture concrete cues and common edge cases that are missing from a short specification.

The fully human-written policy remains strongest overall, but DPR narrows the gap substantially in several domains. Under majority voting, DPR reaches near parity on Misrepresentative, achieving 98% of the human policy performance, and it comes within a small margin on Exploitative. At the same time, Finance Claims remains a clear outlier. This category relies heavily on organization-specific compliance language and fine-grained disclaimer patterns, which are difficult to recover from open web sources alone. These results highlight both the promise and limits of open-domain policy construction. DPR can quickly bootstrap useful draft sections from minimal input, especially in domains where conventions and edge cases are well represented in public guidance, while expert-written rules remain important for categories driven by proprietary standards.

Finally, DPR + Summary reduces the policy length but can trade off accuracy. The compressed policy retains most of the gains on Exploitative and Offensive but loses 1–2 F_1 on Finance Claims and Misrepresentative, suggesting that aggressive compression can remove rare qualifiers that the reader model uses for borderline decisions. We also provide a human evaluation of the generated rules in the original paper's appendix [WLJ26].

6.5 Further Analyses

Alternative Research Model. While the main experiments use Llama 3.3 70B for research, we also ran the pipeline with a smaller Qwen2.5 32B [YYZ24a]. As shown in Figure 6.3 and Figure 6.4, we observe a similar F1 improvement over the baseline, suggesting the generality of the DPR framework.

Convergence Dynamics. Figure 6.2 tracks the number of unique policy rules and the number of keyphrase clusters throughout five research iterations. The curve exhibits a rapid expansion followed by early stabilisation: over 74% of the final rules are discovered in the first iteration, but only 8% further clusters are added after the second. This plateau indicates diminishing returns beyond $k = 3$ iterations and justifies our choice of the research budget.

Indexing Strategy. Figure 6.3 and Figure 6.4 compare three ways of presenting the same policy to the *reader* LLM: (1) a full rule list with the section as the index, (2) a keyphrase *summary* of each cluster, and (3) an ablated "flat" list without section headers. Summaries cut the prompt length but come with a small accuracy drop. The indexed rules and the flattened list have similar performance on Qwen2.5 7B Instruct. However, on Llama 3.1 8B Instruct, the former achieves the best performance. We hypothesize that Llama has a weaker long-context ability and thus can benefit more from more structured input formats.

6.6 Conclusion

Deep Policy Research (DPR) shows that open-domain research agents can autonomously synthesize web-sourced information into effective safety policies. Across online text moderation and multimodal ad review, DPR outperforms definition-only prompts and example-based learning and often recovers nuanced edge cases and policy conventions, demonstrating the feasibility of drafting policies from high-level human guidance.

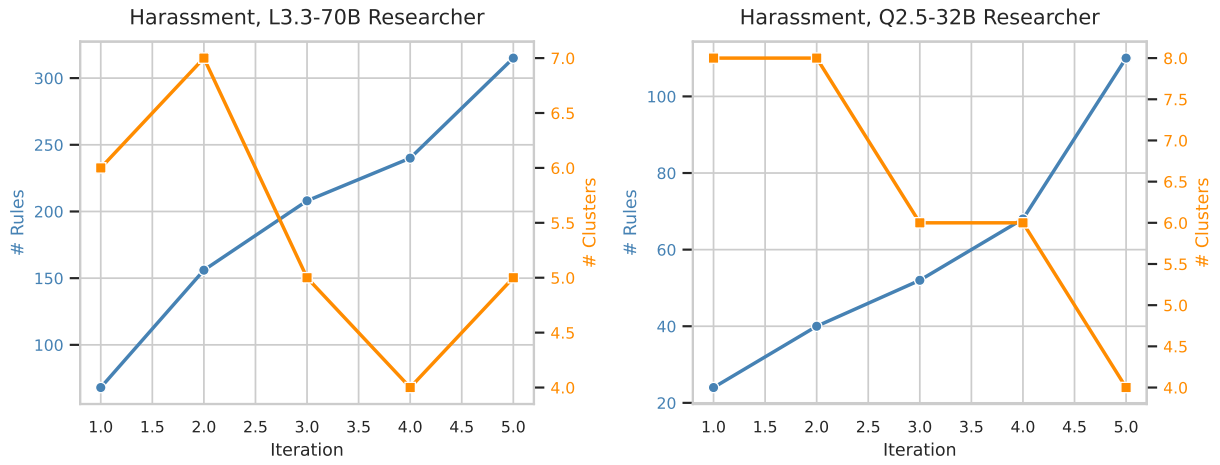


Figure 6.2: Rule count and cluster count vs. iteration in the Harassment domain for DPR initialized with two LLMs. The left panel uses Llama 3.3 70B as the research model and the right uses Qwen2.5 32B. Blue circles show rule counts on the left axis, and orange squares show keyphrase-cluster counts on the right axis.



Figure 6.3: Performance vs. iteration in the Harassment domain with two different LLMs for DPR and Qwen2.5 7B Instruct as the reader. The panels use Llama 3.3 70B and Qwen2.5 32B as the research models. We compare moderation F_1 using a flat rule list, clustered rules, and cluster summaries. The dotted line marks the zero-shot baseline.

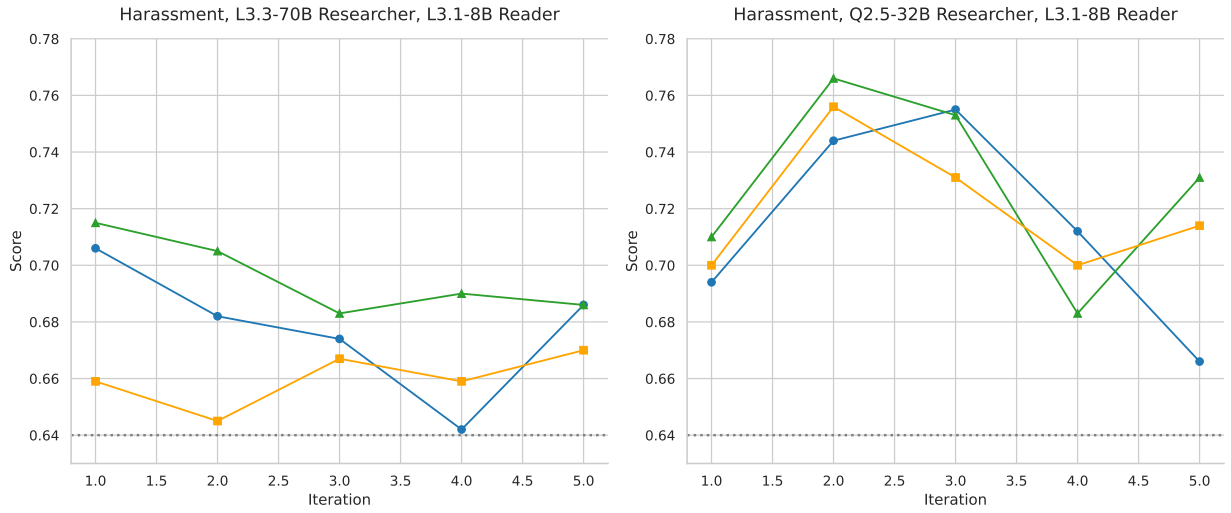


Figure 6.4: Performance vs. iteration in the Harassment domain with two different LLMs for DPR and Llama 3.1 8B Instruct as the reader. The panels use Llama 3.3 70B and Qwen2.5 32B as the research models. We compare moderation F_1 using a flat rule list, clustered rules, and cluster summaries. The dotted line marks the zero-shot baseline.

Limitations Despite these promising results, several directions remain open for future work. First, beyond textual rules, future systems could synthesize representative safe and unsafe examples to provide more actionable guidance. Such example-driven policies may improve downstream interpretability and generalization. Second, while DPR currently relies solely on web content, incorporating human feedback during rule generation or filtering could improve factuality, alignment with organizational values, and trustworthiness. Finally, although we focus on content moderation, DPR's methodology may generalize to broader safety-critical applications such as alignment training data curation, AI deployment policies, or auditing frameworks. We hope our work motivates further exploration into autonomous, scalable policy generation methods that complement human oversight and accelerate the development of safer AI systems.

Potential Risks Our goal is to study an agentic framework (DPR) that drafts safety policies by researching the open web. Even though the downstream task is safety-oriented, the approach could introduce new risks. First, web-sourced material may encode historical

or societal biases. DPR could surface, compress, or over-generalize such biases into policy text, which may propagate into model decisions. Second, policy synthesis can fail subtly (e.g., conflating jurisdictions, misreading sources, or hallucinating "best practices"), which could lead to over-blocking (chilling legitimate speech) or under-blocking (missing harmful content). To mitigate these risks, we recommend that users approach DPR outputs as advisory drafts rather than definitive policy, retain human oversight from relevant reviewers, and plan to revisit decisions as needs, data, and community expectations evolve.

CHAPTER 7

Building a General Procedural Reasoning Datastore

Test-time scaling, where additional inference-time compute is allocated per instance to improve accuracy, has quickly become a central paradigm for frontier reasoning tasks in math, science, and coding [WWS22, WWS23, YYZ23, SLX25, CLS24, LFW25, MYS25]. Recently, reasoning models such as OpenAI's o1 series [Ope24b], DeepSeek R1 [GYZ25], and QwQ [Qwe24] are explicitly trained to produce "thinking" tokens. At inference time, their performance scales up as they are allowed to "think longer" with a larger token budget.

Despite this progress, one limitation of current test-time scaling strategies is that they do not reuse knowledge across reasoning runs. Such knowledge includes both **factual knowledge**, such as definitions, formulas, and theorems, and more importantly **procedural knowledge**: how to reframe a problem, decompose it into subquestions, exploit structure, choose an approach, and verify or backtrack when needed. Discarding procedural knowledge from prior reasoning trajectories can hurt both accuracy and efficiency, as models are forced to repeatedly rediscover useful strategies and revisit unsuccessful lines of thought.

Retrieval-augmented generation (RAG) provides a natural framework for accumulating useful knowledge offline and incorporating it at test time. Recent work has begun to interleave retrieval with reasoning or to introduce general-purpose datastores that improve reasoning performance [LDJ25, SJM25, LDS25]. However, these systems primarily improve access to general background knowledge, while leaving reasoning-specific procedural knowledge implicit. For difficult reasoning tasks, the relevant bottleneck is often not missing facts

alone, but missing guidance on what subquestion to solve next and how to solve it. As a result, retrieved context is often only loosely aligned with the model's current reasoning state. This weak alignment can be especially problematic for reasoning models, whose long reasoning traces may amplify irrelevant contexts. As we later show in our pilot study, standard document-level RAG can yield limited or even negative gains for reasoning models, suggesting the challenge of retrieving knowledge in a form well aligned with the problem and easy for reasoning models to leverage (Section 7.4.1). High-quality procedural knowledge is rare in standard web corpora, and generic retrieval often fails to surface the right subroutine when it is needed.

To address this limitation, we introduce **Reasoning Memory**, a RAG framework that reuses procedural knowledge at scale for reasoning models. As demonstrated in Figure 7.1, rather than indexing generic documents, Reasoning Memory builds a procedural datastore from existing reasoning trajectories. We segment trajectories into atomic subquestion descriptions paired with concise high-level subroutines, yielding a datastore of over 32M items. At inference time, a lightweight in-thought query prompt encourages the model to verbalize its current subquestion as a short query. We retrieve relevant subquestion-subroutine pairs and inject them into the reasoning trace as implicit procedural priors; their source trajectories are not included. We then sample separate groups of reasoning trajectories under diverse retrieved procedures and apply a simple length-based uncertainty heuristic to filter both subroutines and candidate solutions before selecting the final answer. The result is a simple test-time scaling procedure in which additional inference budget enables broader exploration under diverse retrieved procedures, leading to stronger performance as compute increases.

Empirically, Reasoning Memory delivers consistent gains across open-weight reasoning models, inference budgets, and frontier benchmarks in math, science, and coding. It outperforms document-level RAG, trajectory- and template-based procedural RAG, and compute-matched test-time scaling without retrieval. With the higher inference budget, averaged across models and tasks, it improves accuracy by 19.5% over no retrieval and by 8.9% over

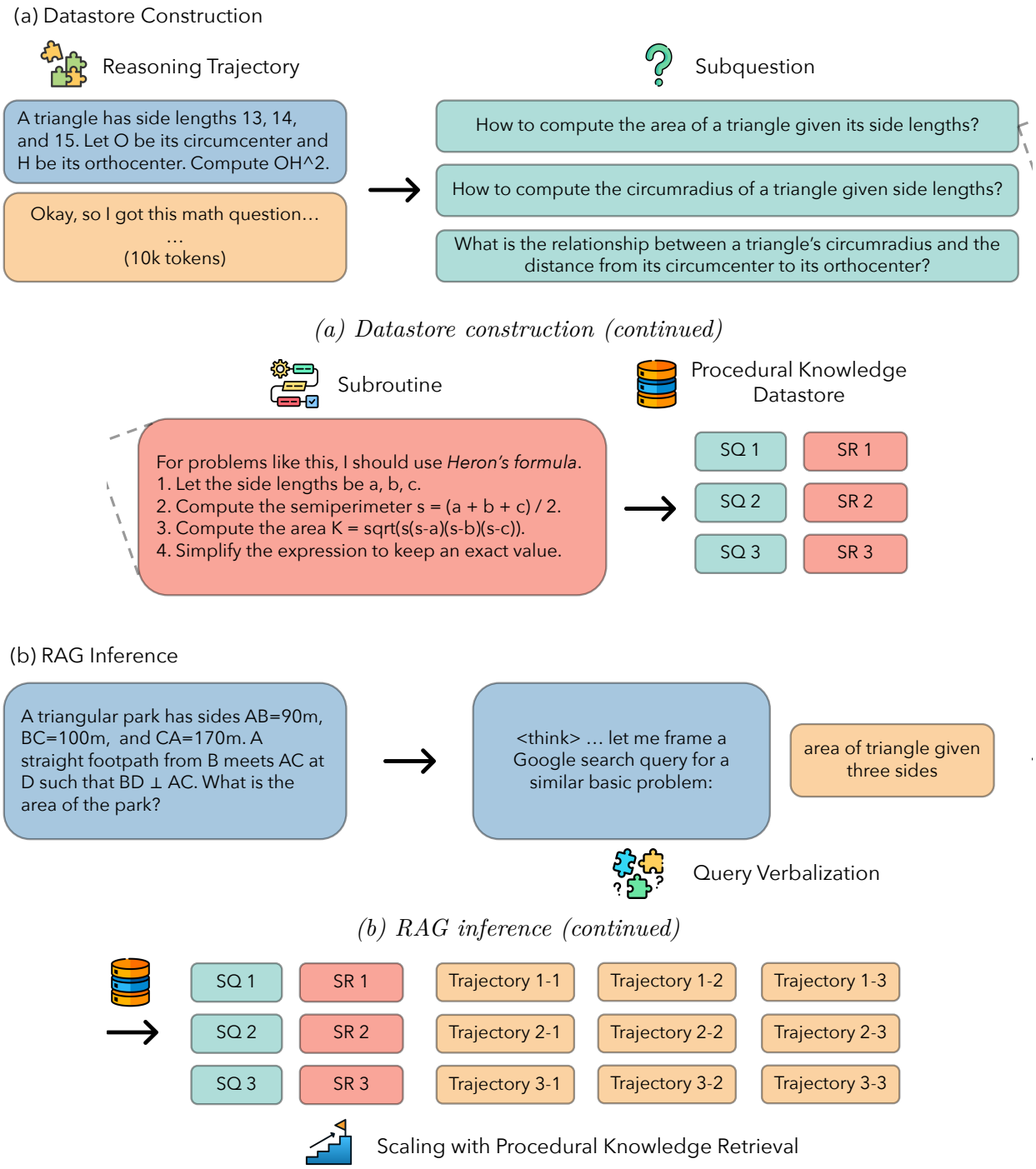


Figure 7.1: **Illustration of the Reasoning Memory framework.** (a) We extract self-contained procedural knowledge from diverse public reasoning trajectories to construct a datastore. (b) At inference time, we retrieve top-k subquestion-subroutine pairs, generate separate groups of candidate trajectories under each subroutine, and filter and rank the candidates. The source trajectories are not inserted.

the compute-matched baseline. Our analyses further show that performance improves with larger and more diverse procedural datastores, and under a fixed inference budget, it is more effective to explore multiple trajectories under a small set of high-quality retrieved subroutines than to spend the same compute under a single subroutine. Additional ablations on datastore construction and in-thought querying identify the contribution of our design.

7.1 Related Work

Test-time Scaling for Reasoning Test-time scaling allocates additional inference-time compute per instance to improve performance on challenging math, science, and coding tasks [WWS22, CLS24, Dee24]. Starting from Chain-of-Thought prompting [WWS22], prior work has explored two broad paradigms. *Sequential scaling* extends a single trajectory through planning, reflection, self-correction, and backtracking [MTG23, CLS24, LFW25, MYS25, XDW25], while *parallel scaling* samples multiple candidates and combines them through selection or aggregation [WWS23, YYZ23, SLX25]. Recent reasoning models explicitly trained to produce long thinking prefixes, including OpenAI's o1 series [Ope24b], DeepSeek R1, and QwQ [GYZ25, Qwe24], exhibit strong inference-time scaling behavior, and simple budget forcing can already yield strong gains on math and science benchmarks [MYS25]. Another line of work combines parallel scaling with search, including process reward models and Monte Carlo Tree Search style methods [WLS24, GNH24, PLG25]. Our work targets reasoning models and augments test-time scaling with a procedural knowledge datastore derived from prior reasoning trajectories. It supports sequential scaling by injecting retrieved procedures into the thinking process and parallel scaling by exploring different retrieved subroutines.

Memorizing Procedural Knowledge A complementary line of work studies how to represent and reuse procedural knowledge such as workflows, strategies, or templates distilled

from experience. Agent Workflow Memory [WMF25] induces reusable workflows from web navigation traces, while ReasoningBank [OYH25] converts successful and failed episodes into compact strategies and supports memory-aware test-time scaling. Think-in-Memory [LYS23] and Dynamic Cheatsheet [SYB25] maintain or adapt memories across interactions and tasks, while Retrieval-of-Thought [AKA25] and trajectory-informed memory [FIJ26] reuse prior thoughts or agent experience. Outside web agents, Buffer of Thoughts [YYZ24b] distills high-level thought templates from problem-solving traces and retrieves them for new instances. Related approaches, including Self-Discover [ZPR24], ReasonFlux [YYC25], and RLAD [QSL25], similarly operate over libraries of heuristics or natural-language abstractions to improve reasoning. Stylistic structure can itself affect reasoning distillation [LY25], so we treat the plan-like form of subroutines as a possible aid rather than attributing gains to style alone. Our contribution is to operationalize procedural memory as a 32M-entry datastore, active in-thought retrieval interface, and compute-matched scaling recipe for reasoning models on math, science, and coding tasks.

RAG for Reasoning Retrieval-augmented generation (RAG) equips language models with external knowledge through retrieval and in-context conditioning [GLT20, LPP20, SMY24]. While most RAG systems are developed for factuality-related tasks, a smaller line of work studies RAG for reasoning-intensive settings. CompactDS builds a compact and diverse web-scale datastore and applies standard retrieval to improve performance on multiple reasoning benchmarks [LDS25]. ReasonIR trains a retriever specialized for reasoning-intensive retrieval [SQK25]. TRACE constructs reasoning chains from retrieved evidence [FMM24], while RAT iteratively revises reasoning steps using retrieved information [WLL24]. Agentic search systems such as Search-o1 [LDJ25] and R1-Searcher [SJM25] instead let models issue queries during reasoning and incorporate retrieved documents. These methods primarily use retrieval to ground reasoning in external information. Reasoning Memory instead retrieves compact procedures distilled from prior reasoning trajectories and supplies them directly to

the thinking process.

7.2 Reasoning Memory: Approach

We introduce **Reasoning Memory**, a RAG framework that induces, retrieves, and reuses *procedural* knowledge at scale for reasoning models. As shown in Figure 7.1, we first distill public reasoning trajectories into a datastore of *subquestion-subroutine* pairs (Section 7.2.1). At inference time, the model verbalizes its current subquestion as a compact query, and the retrieved procedures are injected directly into its thinking stream (Section 7.2.2). We then use multiple retrieved subroutines as implicit priors for test-time scaling (Section 7.2.3).

7.2.1 Procedural Knowledge Datastore

Problem-solving trajectories of current reasoning models contain rich procedural knowledge, but they are too long and noisy to reuse directly. Reasoning Memory therefore converts them offline into a *procedural knowledge datastore* of natural-language subquestions and concise solution subroutines, which supports both effective retrieval and efficient in-context use.

Formally, let a trajectory instance be $(\mathbf{q}, \mathbf{a}, \mathbf{r})$, where $\mathbf{q}$ is the original problem, $\mathbf{a}$ is the final answer, and $\mathbf{r}$ is the intermediate reasoning trace. We map $(\mathbf{q}, \mathbf{r})$ to a set of K subquestion-subroutine pairs $\{(q_i, s_i)\}_{i=1}^{K}$ using a two-step prompting pipeline. First, we derive self-contained subquestions $\{q_i\}$ that capture the key intermediate goals in the trajectory. Then, for each q_i, we generate a concise subroutine s_i that summarizes the high-level procedure used to address it, while abstracting away local calculations and incidental trial-and-error. We do not enforce a single abstraction level: the datastore contains both broad strategies and local procedures to match subquestions of different granularity. This yields a reusable representation that remains short enough to serve as an in-context procedural prior.

Rather than collecting trajectories from scratch, we build the datastore from the publicly released Nemotron V1 corpus [NGM25], which covers diverse math, science, and coding ques-

tions. The resulting datastore contains approximately 32 million (q_i, s_i) pairs. On average, subquestions contain 19.2 tokens, subroutines 207.9 tokens, and each trajectory contributes 10.5 subquestions. The full datastore construction details are presented in the original paper's appendix [WSY26]. QwQ-32B is used for both prompting steps[1].

Source Corpus and Deduplication We use the math, stem, and code subsets, which contain 2.0, 20.7, and 1.9 million trajectories, respectively. Nemotron pairs each question with multiple reasoning trajectories generated with a teacher model. Empirically, we found that these trajectories often converge to similar general approaches and differ mostly in local steps or formatting. As such, decomposing a large number of trajectories for each question would bring marginal gains in datastore diversity. We thus select the first trajectory per question to construct the datastore. After this deduplication process, we obtained roughly 170k unique questions for math, 3 million for stem, and 34k for code, comprising approximately 10% of the original dataset. Each trajectory contains a user question and a response generated by DeepSeek-R1.

7.2.2 In-Thought Active Retrieval Augmentation

Given a large datastore of self-contained problem-procedure pairs, the next challenge is querying it effectively to benefit a reasoning model's thinking process. To begin with, prompting the model to follow certain query generation formats within its thoughts is challenging. On the other hand, approaches such as Search-o1 [LDJ25] that defer query generation until extensive reasoning is expensive and is better suited factuality-related gaps than reasoning strategies which require early interventions to model thinking.

Inspired by Jiang et al. [JXG23], we instead leverage a simple *thought-hijacking* prompt to enable query generation part of the thinking stream itself. Concretely, we start the model's

[1]Smaller models are also effective, as most of the knowledge is already in the trajectories (Table 7.2).

thinking with the following meta-thinking sentence:

`Now, let me search for a similar basic problem whose solution can help unblock me for solving the current step. Let me frame it as a more high-level google search query:`

We extract the next sentence in the continuation as the retrieval query $\tilde{q}$. Here, "search query" denotes a short search-style verbalization rather than a keyword-only query. It identifies the initial subproblem and removes incidental details such as long specifications, ad-hoc entities, symbols, examples, and formatting constraints. In practice, this produces concise subquestion descriptions that align well with datastore subquestions. Given $\tilde{q}$, we retrieve the top-k subquestions and their corresponding subroutines, denoted by $\{(\hat{q}_j, \hat{s}_j)\}_{j=1}^{k}$, using a standard dense retriever. The subquestion acts as a concise key for its associated procedure; matching a question-like query to this key is more direct than matching it to procedural prose. The original source trajectories are not inserted at inference time. For each retrieved pair, we insert the following hint directly into the reasoning stream:

`[hint] Here is a problem solving procedure for a related question "`$\hat{q}_j$`": `$\hat{s}_j$` [end of hint]`

This hint is followed by a simple continuation cue, "`Okay,`". The model then continues reasoning conditioned on the retrieved procedural prior. Because both the query and the retrieved subroutine appear in the same thinking channel, retrieval acts as a lightweight extension of the model's ongoing reasoning.

7.2.3 Inference-Time Scaling with Reasoning Memory

Finally, Reasoning Memory introduces a test-time scaling method that naturally adapts to different compute budgets (in terms of samples) by adjusting the diversity of retrieved procedures and intensity of per-procedure compute. Given a sampling budget of at most m samples

and top-k retrieved subroutines $\{(\hat{q}_j, \hat{s}_j)\}_{j=1}^{k}$, we allocate $\lfloor m/k \rfloor$ samples to each retrieved subroutine using the retrieval-augmented prompt from Section 7.2.2. Each subroutine conditions a separate group of newly generated trajectories rather than being concatenated with the other retrieved subroutines. This yields a pool of candidate trajectories $\{\pi_{j,\ell}\}$, where $j \in \{1, \ldots, k\}$ indexes the retrieved subroutine and $\ell \in \{1, \ldots, \lfloor m/k \rfloor\}$ indexes samples under that subroutine.

We then score each trajectory $\pi_{j,\ell}$ with an uncertainty measure $r_{j,\ell}$ and normalize scores across the full pool:

$$\tilde{r}_{j,\ell} \equiv \frac{\max_{j',\ell'} r_{j',\ell'} - r_{j,\ell}}{\max_{j',\ell'} r_{j',\ell'} - \min_{j',\ell'} r_{j',\ell'}} .$$

For each retrieved subroutine, we compute an average quality score

$$\bar{r}_j \equiv \frac{1}{\lfloor m/k \rfloor} \sum_{\ell=1}^{\lfloor m/k \rfloor} \tilde{r}_{j,\ell} .$$

We retain subroutines with $\bar{r}_j > \tau$, rank their associated trajectories by confidence, and keep the top n samples for pass@1-based evaluation. If fewer than n samples remain, we backfill with the highest-confidence samples from the filtered-out pool.

Inspired by Hassid et al. [HSA25], we use the thinking length as the default uncertainty score. The intuition is that uncertain reasoning tends to produce longer traces due to extra branching and backtracking. Concretely, we set $r_{j,\ell}$ to the generated trajectory length in tokens, so that after normalization, larger $\tilde{r}_{j,\ell}$ indicates a shorter and potentially more confident trajectory relative to the rest of the pool. The subroutine score $\bar{r}_j$ therefore measures whether a retrieved subroutine tends to induce shorter samples. By design, the pipeline is compatible with other uncertainty signals, such as log-likelihood, entropy, or self-evaluated relevance. Nevertheless, we find length to be a strong and simple choice among these alternatives and use it for the main results. Length is only a heuristic confidence proxy: verbosity differences and premature convergence can break the correlation between shorter traces and correctness. Averaging across samples for each subroutine reduces reliance on any single short trajectory, but does not eliminate mismatched retrievals (the original paper's appendix [WSY26]). We

provide a full comparison in Section 7.4.8.

7.3 Experimental Setup

Models We focus on open-weight reasoning models that expose their thinking tokens. Our main evaluation uses three models spanning different sizes, base model families, and post-training recipes: DeepSeek-R1-Distill-Llama-8B [GYZ25], OpenThinker3-7B [GMK25], and Qwen3-32B [YLY25]. Each model is used in the reasoning mode without additional fine-tuning.

Benchmarks and Metrics We evaluate on challenging math, science, and coding benchmarks. For math, we use AIME 2024 and AIME 2025 (30 problems each) [Mat24], and MATH500 [LKB24], a standard subset of the MATH test set [HBK21]. For science, we use GPQA-Diamond (GPQA-D) [RHS24]. For coding, we use LiveCodeBench (LCB) [JHG25]; following the filtering criteria of Li et al. [LDJ25], we evaluate on 112 problems from releases V1–V4 and 109 problems from releases V5–V6. We use standard metrics from prior work: math-equal for AIME and MATH500, exact match for GPQA-D, and execution-based pass@1 for LiveCodeBench, using the implementations of Li et al. [LDJ25]. Unless otherwise specified, we sample $m = 8$ trajectories per problem and report the averaged performance (equivalent to pass@1). For methods with uncertainty-based selection, the performance is averaged over the selected n samples after filtering. Our main setting uses $(m, n) = (8, 4)$. We also evaluate $(m, n) = (30, 8)$ and analyze larger budgets up to $m = 100$ in Section 7.4.4.

Baselines We compare Reasoning Memory with baselines covering RAG with trajectory-level knowledge, template-based procedural knowledge, document-level factual knowledge, as well as retrieval-free uncertainty-based test-time scaling.

- **No RAG.** The model solves each problem without retrieval.

- **Trajectory RAG.** We retrieve reasoning trajectories from the Nemotron v1 corpus using the question as the query, and either (1) prepend the first 256 tokens of each of the top 3 trajectories (*prefix*) or (2) compress the top trajectory into a short summary and prepend it (*summary*). This baseline uses the same source corpus as Reasoning Memory, but operates at the trajectory level rather than the subroutine level.

- **Template RAG.** We instantiate RAG over small human-designed or automatically distilled template libraries from prior work. The *ReasonFlux* variant uses templates from Yang et al. [YYC25], and the *Self-Discover* variant uses templates from Zhou et al. [ZPR24]. For fairness, all retrieved reasoning templates are injected into the thinking stream in the same way as in Reasoning Memory.

- **Document RAG.** We include two factual knowledge retrieval baselines: retrieving from *CompactDS* following Lyu et al. [LDS25], and retrieving from Google[2]. In both cases, the original question is used as the query and the retrieved passages are prepended to the question in the prompt, following Li et al. [LDJ25].

- **Length Scaling.** A retrieval-free test-time scaling baseline inspired by Hassid et al. [HSA25]. We sample m independent trajectories and select n high-confidence ones using the length-based uncertainty heuristic from Section 7.2.3. This isolates the benefit of procedural retrieval[3]. We evaluate $(m, n) = (8, 4)$ and $(30, 8)$.

- **Reasoning Memory.** Our full method retrieves k procedural subroutines from the Nemotron-based datastore, injects them in thought using the thought-hijacking prompt, and performs two-stage length-based filtering over m trajectories. We use

[2]We set the search date cutoff to 2023/12/31 to prevent data contamination.

[3]In the main text, we do not apply length scaling to the other baselines, since this is not standard practice. When added, the relative performance trends remain similar. Results are reported in the original paper's appendix [WSY26].

$\tau = 0.1$ and evaluate $(m, n, k) = (8, 4, 3)$ or $(30, 8, 10)$, carefully matching the sampling budget of the Length Scaling baseline.

For all retrieval-based methods, we use ReasonIR-8B [SQK25] as the retriever. Full implementation and efficiency details are provided in the original paper's appendix [WSY26].

7.4 Results

7.4.1 Standard Document RAG is Poorly Aligned with Reasoning Models

We begin with a pilot study to test whether a standard document-level RAG pipeline benefits reasoning models in the same way it has been shown to benefit instruction-tuned models. CompactDS reported that a web-scale general document datastore can improve instruction-tuned models on challenging reasoning benchmarks, but did not evaluate this setup on reasoning models [LDS25]. We therefore follow the CompactDS setup and compare paired instruction-tuned and reasoning models from the same families on AIME 2024, GPQA-Diamond, and LiveCodeBench. Full setup details are presented in the original paper's appendix [WSY26].

Figure 7.2 shows a consistent pattern across model families and tasks: instruction-tuned models benefit modestly from retrieved documents, while the corresponding reasoning models often gain little or even degrade. This suggests that standard document-level RAG is poorly aligned with reasoning models, whose current subquestions often require procedural guidance rather than generic background context. In Section 7.4.2, we further diagnose this phenomenon using synthesized knowledge. The results show that reducing retrieval noise makes injected knowledge more effective, and that the *type* of knowledge also matters: procedural guidance yields larger gains than factual knowledge for reasoning models. This distinction is functional rather than absolute: a subroutine may include a fact, theorem, or formula, but explains how to apply and verify it within a solution process. This motivates

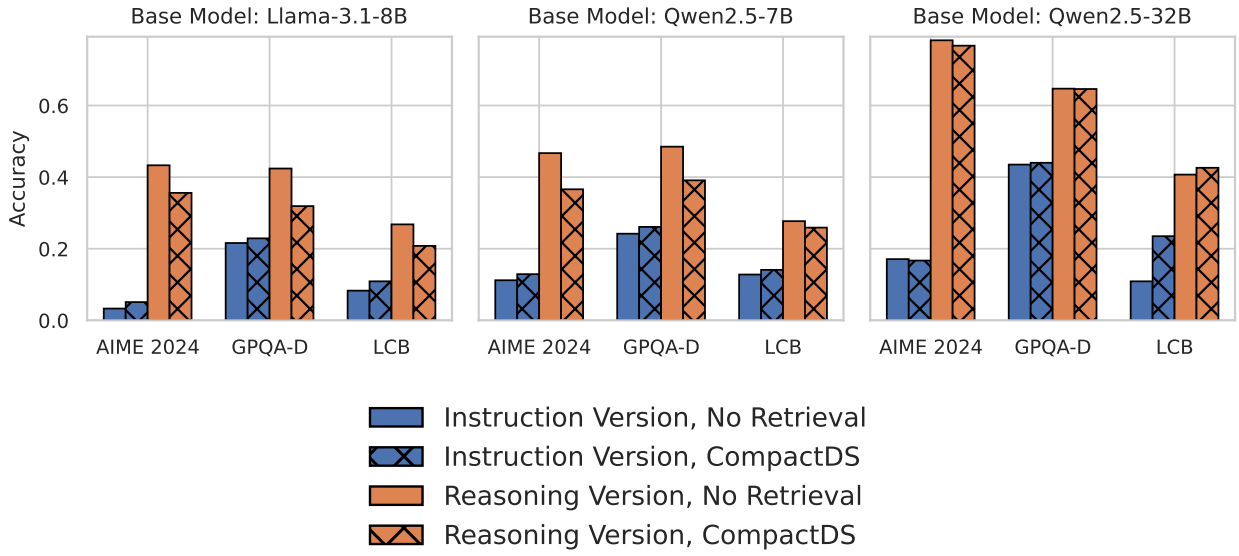


Figure 7.2: **Standard document RAG benefits instruction-tuned models more than reasoning models.** Under the CompactDS pipeline, instruction-tuned models obtain modest gains from retrieval, whereas the corresponding reasoning models often see limited gains or even degradation, despite much stronger no-retrieval performance.

our focus on compact procedural subroutines rather than general documents.

7.4.2 Controlled Knowledge Injection: Factual vs. Procedural

The CompactDS result above leaves open an important question: are reasoning models inherently incompatible with retrieval, or is the issue the form of the retrieved knowledge? To probe this question, we run a controlled knowledge injection study in which the retrieved content is synthesized to be directly relevant to each evaluation problem.

For each problem from AIME 2024, GPQA-Diamond, and LiveCodeBench, we prompt a strong model (GPT-5.2 via API, OpenAI [Ope25a]) to synthesize a short paragraph that is helpful for solving the problem while satisfying two constraints: (1) it must not reveal the final answer, and (2) it must not provide intermediate computations or step-by-step derivations that could leak the answer. We synthesize two types of content: (1) *factual knowledge*, including definitions, formulas, and theorems, and (2) *procedural knowledge*, in-

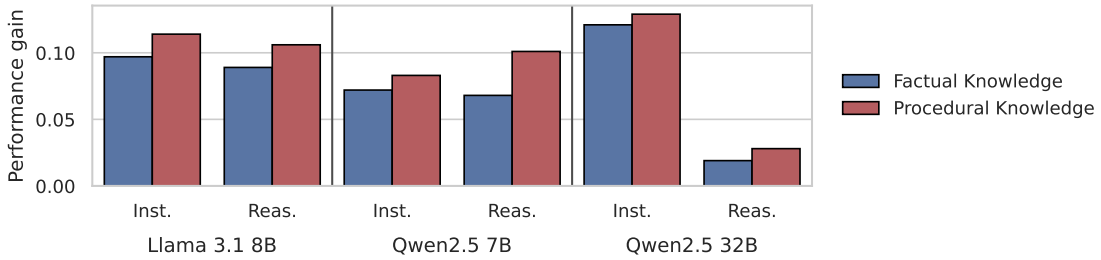


Figure 7.3: **Utility of different types of synthesized knowledge.** We report performance gains relative to the no-retrieval setting. Both factual and procedural knowledge help, but procedural knowledge yields larger gains across tasks and model families.

cluding high-level decomposition strategies, solution plans, and verification heuristics. The resulting paragraph is typically 100–200 tokens long and is inserted into the same prompt location used for CompactDS documents.

Figure 7.3 reports performance deltas from the no-retrieval setting. Both instruction-tuned and reasoning models improve when given synthesized knowledge, which suggests that retrieval itself is not the core issue. Instead, the usefulness of retrieval depends strongly on the form and alignment of the injected content. In particular, reasoning models benefit much more from these targeted knowledge paragraphs than from retrieved CompactDS documents, even though both are inserted through the same prompt template. This supports the view that reasoning models are sensitive to the relevance and specificity of the injected context.

Procedural knowledge is consistently more helpful than factual knowledge across tasks and model families. We hypothesize that this is because many difficult reasoning problems are bottlenecked less by missing background facts than by missing guidance on how to decompose the problem, choose a strategy, or verify an intermediate direction. This pattern is especially natural for AIME and LiveCodeBench, which often require problem reframing and multi-step inference. GPQA-Diamond benefits somewhat more from factual knowledge, likely because some questions can be answered more directly from domain concepts or definitions, but procedural guidance remains stronger overall.

These results motivate our focus on building a scalable datastore that explicitly captures

procedural knowledge and retrieves it in a form that reasoning models can readily use.

7.4.3 Main Results

Table 7.1 summarizes the end-to-end accuracy of Reasoning Memory and all baselines. Most baselines fail to consistently outperform No RAG on frontier reasoning tasks. Averaged over all tasks, only Google-based Document RAG improves over No RAG for two models: DeepSeek-R1-Distill-Llama-8B (0.422 vs. 0.413) and Qwen3-32B (0.661 vs. 0.634). In contrast, Reasoning Memory consistently improves over No RAG by a substantial margin, reaching 0.580 versus 0.500 when averaged across all models and tasks. The gains span the evaluated 7–8B models and Qwen3-32B, with the largest improvement on OpenThinker3-7B. This shows that, under a simple RAG pipeline, reasoning models benefit much more from large-scale procedural retrieval than from web documents, full trajectories, or small template libraries.

Reasoning Memory also remains stronger than the retrieval-free Length Scaling baseline in most settings, winning 34 out of 36 comparisons across models, tasks, and the two budget settings $m \in \{8, 30\}$. Increasing the inference budget further improves performance. At $m = 30$, Reasoning Memory improves over No RAG by 0.12 (19.0%) on math, 0.05 (9.8%) on science, and 0.09 (28.9%) on coding, averaged across models. These gains support procedural knowledge retrieval as an effective way to improve test-time scaling for reasoning models. We next examine this effect in more detail and show that the gap remains consistent as the inference budget increases and more samples are drawn. In the later sections, we also analyze the datastore composition and key design ablations in detail. Additional empty-context and majority-vote controls are reported in Section 7.4.7, and qualitative examples are provided in the original paper's appendix [WSY26].

Method	Ret.	m	Math			Science	Code (LCB)	
			AIME 24'	AIME 25'	MATH 500	GPQA-D	V1-4	V5-6
DeepSeek-R1-Distill-Llama-8B								
No RAG	N/A	8	0.461	0.350	0.806	0.427	0.220	0.211
Trajectory RAG (Prefix)	Proc.	8	0.425	0.246	0.784	0.391	0.195	0.128
Trajectory RAG (Summary)	Proc.	8	0.408	0.321	0.822	0.431	0.184	0.121
Template RAG (ReasonFlux)	Proc.	8	0.450	0.342	0.828	0.417	0.174	0.210
Template RAG (Self-Discover)	Proc.	8	0.442	0.296	0.821	0.414	0.213	0.247
Document RAG (CompactDS)	Fact.	8	0.356	0.312	0.818	0.319	0.208	0.172
Document RAG (Google)	Fact.	8	0.483	0.351	**0.838**	0.383	0.246	0.228
Length Scaling	N/A	8	0.508	0.325	0.815	0.426	0.255	0.278
Reasoning Memory	Proc.	8	**0.511**	**0.364**	**0.838**	**0.444**	**0.278**✣	**0.292**✣
Length Scaling	N/A	30	0.548	0.358	0.802	0.447	0.282	0.302
Reasoning Memory	Proc.	30	**0.575**	**0.392**✣	**0.836**✣	**0.461**	**0.310**✣	**0.325**✣
OpenThinker3-7B								
No RAG	N/A	8	0.470	0.392	0.841	0.438	0.318	0.266
Trajectory RAG (Prefix)	Proc.	8	0.429	0.358	0.828	0.437	0.242	0.249
Trajectory RAG (Summary)	Proc.	8	0.438	0.354	0.855	0.451	0.242	0.260
Template RAG (ReasonFlux)	Proc.	8	0.467	0.354	0.830	0.457	0.262	0.268
Template RAG (Self-Discover)	Proc.	8	0.442	0.392	0.825	0.441	0.276	0.279
Document RAG (CompactDS)	Fact.	8	0.454	0.333	0.846	0.458	0.292	0.258
Document RAG (Google)	Fact.	8	0.501	0.422	0.864	0.452	0.235	0.231
Length Scaling	N/A	8	0.612	0.506	0.876	0.484	0.333	0.344
Reasoning Memory	Proc.	8	**0.725**✣	**0.646**✣	**0.908**✣	**0.522**✣	**0.362**✣	**0.397**✣
Length Scaling	N/A	30	0.647	0.528	0.873	0.502	0.345	0.343
Reasoning Memory	Proc.	30	**0.758**✣	**0.679**✣	**0.911**✣	**0.542**✣	**0.381**✣	**0.412**✣
Qwen3-32B								
No RAG	N/A	8	0.789	0.586	0.905	0.669	0.387	0.465
Trajectory RAG (Prefix)	Proc.	8	0.683	0.408	0.895	0.607	0.249	0.432
Trajectory RAG (Summary)	Proc.	8	0.629	0.429	0.895	0.595	0.267	0.418
Template RAG (ReasonFlux)	Proc.	8	0.654	0.500	0.901	0.644	0.300	0.278
Template RAG (Self-Discover)	Proc.	8	0.667	0.483	0.903	0.633	0.312	0.339
Document RAG (CompactDS)	Fact.	8	0.604	0.388	0.899	0.602	0.267	0.284
Document RAG (Google)	Fact.	8	0.795	**0.767**	0.919	0.678	0.382	0.426
Length Scaling	N/A	8	0.792	0.591	0.903	**0.699**	0.399	0.444
Reasoning Memory	Proc.	8	**0.825**✣	**0.767**	**0.926**✣	0.674	**0.462**✣	**0.503**✣
Length Scaling	N/A	30	0.812	0.619	0.908	**0.682**	0.405	0.476
Reasoning Memory	Proc.	30	**0.838**	**0.754**✣	**0.923**✣	0.681	**0.471**✣	**0.508**✣

Table 7.1: **Main results of all methods in math, science, and coding domains.** *Ret.* = type of retrieved knowledge, Proc. = procedural, Fact. = factual. Bold numbers mark the best performance under the same budget. ✣ = statistically significantly better than the second best via paired t-test ($p < 0.05$). Reasoning Memory significantly outperforms baselines across LLMs and generally improves over Length Scaling across budgets.

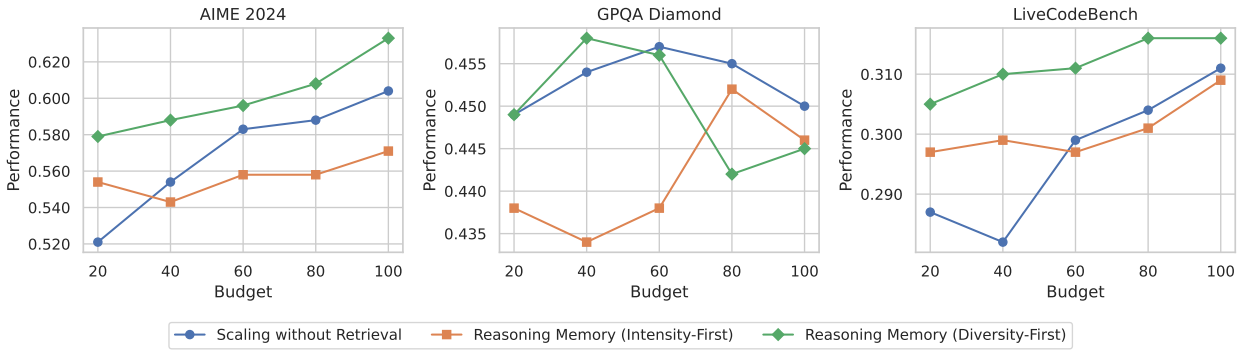


Figure 7.4: **Performance as a function of inference budget.** We compare Length Scaling without retrieval with two Reasoning Memory variants on DeepSeek-R1-Distill-Llama-8B as the total sampling budget m increases. Intensity-First allocates 20 samples to each selected subroutine, while Diversity-First distributes the budget across the top 20 retrieved subroutines. The panels report accuracy on AIME 2024, GPQA-D, and LiveCodeBench.

7.4.4 Scaling Behavior with Larger Inference Budgets

We next study whether Reasoning Memory continues to help as the inference budget increases, and how the budget should be allocated across retrieved subroutines. Using DeepSeek-R1-Distill-Llama-8B, we evaluate $m \in \{20, 40, 60, 80, 100\}$ under three strategies: (1) *Length Scaling*, which samples m trajectories without retrieval; (2) *Reasoning Memory (Intensity-First)*, which allocates 20 samples to each retrieved subroutine and uses only a subset of the top-ranked subroutines to stay within the budget limit; and (3) *Reasoning Memory (Diversity-First)*, which always uses the top 20 retrieved subroutines and allocates the maximum number of samples per subroutine within the budget limit.

As shown in Figure 7.4, the diversity-first strategy yields the strongest scaling behavior. On AIME 2024 and LiveCodeBench (V1–V4), it improves monotonically with budget and achieves the best performance at $m = 100$, clearly outperforming both Length Scaling and intensity-first Reasoning Memory. On GPQA-D, all methods show weaker returns from additional budget, but diversity-first Reasoning Memory remains competitive with Length Scaling, and its best result is still achieved at a lower budget[4]. Overall, these results highlight

[4]In fact, in Table 7.1, the main setting reaches an even higher GPQA-D score of 0.461 with $m = 30$.

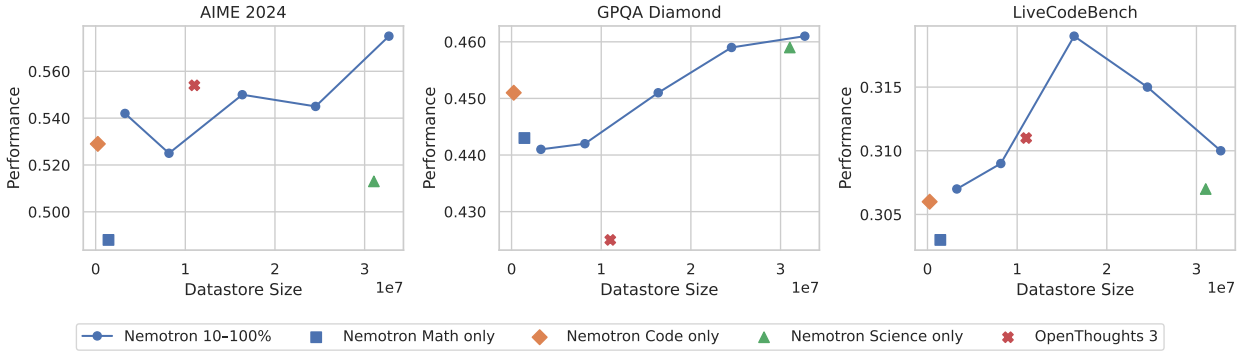


Figure 7.5: **Effect of datastore size and composition on Reasoning Memory.** Performance of DeepSeek-R1-Distill-Llama-8B with budget $m = 30$. Larger and more diverse datastores generally yield stronger performance. The connected curve uses random subsets spanning 10% to 100% of Nemotron trajectories. Separate markers show math-only, code-only, and science-only Nemotron subsets and an OpenThoughts3 datastore.

the importance of a diversity-first budget allocation strategy and confirm the advantage of Reasoning Memory at larger budgets.

7.4.5 Impact of Datastore Composition

We next study how the size and composition of the procedural datastore affect Reasoning Memory. Using DeepSeek-R1-Distill-Llama-8B with $m = 30$, we compare three classes of datastore variants: (1) random subsets of Nemotron ranging from 10% to 100% of trajectories, (2) domain-specific Nemotron subsets containing only math, code, or science trajectories, and (3) a datastore built from OpenThoughts3, which is more math-focused.

Figure 7.5 shows that performance generally improves as we scale the mixed Nemotron datastore. The full datastore gives the best accuracy on AIME 2024 and GPQA-D, while LiveCodeBench is less sensitive once the datastore reaches roughly 25% of full scale. Among domain-specific datastores, OpenThoughts3 is highly competitive on AIME 2024, matches similarly sized Nemotron subsets on LiveCodeBench, but underperforms clearly on GPQA-D. Within Nemotron, science-only trajectories give the best science performance on GPQA-D, while code-only trajectories are surprisingly more effective than math-only trajectories across

Setting	AIME 2024	GPQA-D	LCB (V1-4)	Average
Removing Subroutine Decomposition	0.551	0.433	0.308	0.431
Removing Self-Generated Query	0.567	0.453	0.295	0.438
Full Reasoning Memory (Qwen3-8B)	0.550	**0.474**	**0.320**	0.448
Full Reasoning Memory (QwQ-32B)	**0.575**	0.461	0.310	**0.449**

Table 7.2: **Ablations on datastore construction and query design.** All results use DeepSeek-R1-Distill-Llama-8B with budget $m = 30$. The full Reasoning Memory configuration uses QwQ-32B as the datastore generator, a decomposed datastore indexed by subquestions, and self-verbalized in-thought queries.

all three tasks despite having similar size. Code trajectories may transfer through domain-general patterns such as decomposition, constraint tracking, case analysis, debugging, and verification. For both math and coding, however, all in-domain subsets underperform the full mixed datastore, suggesting that broader procedural coverage is useful. These results do not identify which entries, or what fraction of the full datastore, drive the gains.

7.4.6 Datastore and Query Ablations

We ablate three key design choices of Reasoning Memory using DeepSeek-R1-Distill-Llama-8B with $m = 30$. First, we remove the subquestion decomposition and instead index the datastore by the original question, with a single subroutine summarizing the full trajectory. Second, we keep the decomposed datastore but disable self-verbalized retrieval queries, using the original question directly as the query. Third, we replace QwQ-32B with the smaller Qwen3-8B for datastore generation. Finally, the uncertainty criterion is another important design choice, which we study in Section 7.4.8 due to space constraints.

As shown in Table 7.2, Reasoning Memory is robust to the choice of the datastore generator model, but both decomposition and self-verbalized queries matter. Using Qwen3-8B yields performance close to the full system, and even slightly improves GPQA-D and LiveCodeBench, suggesting that much of the useful procedural knowledge already resides in the source trajectories. In contrast, removing subroutine decomposition causes the largest drop on AIME and GPQA-D, indicating that fine-grained decomposition is important for

building a diverse datastore that matches different subquestion granularities. Replacing self-verbalized queries with the original question also leads to consistent degradation across all three benchmarks. The largest drop is seen on LiveCodeBench, where problem statements are often long and cluttered with context, format specifications, and examples. Since AIME 2024 contains only 30 problems, small single-task differences should not be overinterpreted; our component conclusions rely on the combined pattern across AIME 2024, GPQA-D, and LiveCodeBench. The stronger end-to-end evidence in Table 7.1 spans three models, six benchmarks, and two sampling budgets, with significance markers reported. Overall, these ablations support the core design choices of Reasoning Memory while suggesting that datastore construction can be performed with smaller models at little cost.

Taken together, our experiments examine four main design axes of a RAG-style memory system: the retrieved value, through documents, trajectories, summaries, templates, and subroutines (Table 7.1); the key and query, through full-question indexing and original-question queries (Table 7.2); and utilization and selection, through budget allocation (Figure 7.4) and alternative uncertainty criteria (Table 7.4).

7.4.7 Empty-Context and Majority-Vote Controls

To isolate the contribution of retrieval from query verbalization, we replace the retrieved subroutine with an empty context while retaining query verbalization and length-based selection. We also compare against majority voting on AIME 2024 and GPQA-D, where answer aggregation is well defined. As shown in Table 7.3, Reasoning Memory is strongest in three of the four model-task comparisons, while majority voting is stronger for DeepSeek-R1-Distill-Llama-8B on GPQA-D. Empty context is consistently weaker than Reasoning Memory on AIME and does not explain its overall gains.

Model	Method	AIME 2024	GPQA-D
OpenThinker3-7B	Length Scaling	0.647	0.502
	Empty Context + Length Scaling	0.547	0.511
	Majority Vote	0.573	0.533
	Reasoning Memory	**0.758**	**0.542**
DeepSeek-R1-Distill-Llama-8B	Length Scaling	0.548	0.447
	Empty Context + Length Scaling	0.555	0.452
	Majority Vote	0.496	**0.472**
	Reasoning Memory	**0.575**	0.461

Table 7.3: **Empty-context and majority-vote controls.** All results use $m = 30$. Empty Context retains query verbalization but provides no retrieved subroutine before applying length-based selection. Majority voting is reported only for AIME 2024 and GPQA-D, where answer aggregation is well defined.

7.4.8 Uncertainty Criteria

Our main system uses generation length as the uncertainty proxy for stage 1 item-level filtering after retrieval. In this section, we compare this metric with several alternative criteria while keeping stage 2 fixed to length-based selection. Specifically, for each retrieved subroutine, we compute:

- **Likelihood**: log likelihood of the first 200 continued tokens conditioned on the retrieved subroutine. For this metric, lower is better as we expect the retrieval to inject new and unfamiliar information.
- **Entropy**: average token-level entropy of the first 200 continued tokens conditioned on the retrieved subroutine. Higher is better.
- **Contrast**: the distributional difference between the downstream reasoning model and its base pretrained model, evaluated on the first 200 tokens generated in the continuation. This metric is inspired by Ouyang et al. [OZX25] and we instantiate it with KL Divergence (higher is better).
- **Self-Eval**: a scalar score where the model is prompted to rate how relevant the retrieved subroutine is to the problem on a Likert scale. We repeatedly sample for 10

Metric	AIME 24'	GPQA-D	LCB (V1–4)
Likelihood	0.571	0.429	0.273
Entropy	**0.600**	0.417	0.273
Contrast	0.563	0.419	0.269
Self-Eval	0.571	0.402	0.263
Length	0.575	**0.461**	**0.310**

Table 7.4: **Alternative metrics for step 1 filtering.** Results for DeepSeek-R1-Distill-Llama-8B with budget m=30; stage 2 always uses length as the uncertainty proxy. Length achieves strong overall performance while being the simplest.

times with temperature = 0.6 and take the average as the score to apply the filter.

All scores are then normalized using the same procedure as in Section 7.2.3. Using DeepSeek-R1-Distill-Llama-8B with a budget of $m = 30$, we report the task accuracy results in Table 7.4. Overall, no single alternative dominates across all benchmarks, but length is the most reliable choice overall. On AIME 24', entropy achieves the highest score (0.600), with length close behind at 0.575. On GPQA-D and LCB (V1–4), however, length clearly outperforms all other metrics (0.461 vs. 0.429 or lower on GPQA-D, and 0.310 vs. at most 0.273 on LCB). Averaged over the three tasks, length is therefore the strongest option, while also being significantly simpler and cheaper than metrics that require extra model evaluations. For this reason we adopt length as our default uncertainty proxy in the main experiments.

7.4.9 Retrieval vs. Training

Reasoning Memory provides a model-agnostic way to digest a reasoning corpus as an external datastore and perform RAG at inference time. Alternatively, post-training can internalize the corpus into a model's parameters. Is Reasoning Memory still useful after the same corpus has been internalized? We investigate this question using the Llama-3.1-Nemotron-Nano-8B-v1 model released by Nvidia. According to the model card, this model was likely trained on variants of the Nemotron v1 corpus via supervised fine-tuning (SFT) and reinforcement learning (RL). We emphasize that this comparison is not fully controlled for two reasons:

Method	AIME 24'	GPQA	LCB (V1–4)
No RAG	0.658	**0.553**	0.380
Length Scaling	0.692	0.548	**0.396**
Reasoning Memory	**0.708**	0.534	0.395

Table 7.5: Results on AIME 24', GPQA, and LCB (V1–4) using Llama-3.1-Nemotron-Nano-8B-v1. For both Length Scaling and Reasoning Memory we use the budget $m = 30$ and $n = 8$.

(1) the detailed training data composition and compute recipe are not entirely clear, and (2) due to question deduplication, we use a Nemotron subset approximately 10% as large as the original corpus. A fully controlled comparison would re-run post-training from the backbone model, which is beyond our budget.

As shown in Table 7.5, we observe mixed results when comparing Reasoning Memory with No RAG and Length Scaling. Although the model was trained on a similar corpus, Reasoning Memory still improves over No RAG on AIME and LCB. However, it does not outperform Length Scaling on GPQA or LCB. These results indicate that fine-tuning is a strong way to learn from a corpus and that the contribution of Reasoning Memory may depend on how much of the corpus the model has already internalized. This also opens several future directions, such as expanding the knowledge sources, actively learning the memory corpus, and learning subroutines that complement the model's existing abilities.

7.5 Conclusion

Reasoning Memory shows that test-time compute for reasoning models can be spent not only on producing longer chains of thought, but also on reusing prior problem-solving experience in a structured way. By surfacing procedural knowledge from existing reasoning trajectories and turning it into an explicit datastore, our approach transforms instruction-tuning corpora into an inference-time resource for guiding new problem-solving episodes. We further introduce a simple recipe for aligning retrieval with the model's own thinking stream and for

scaling inference with multiple retrieved priors. Empirically, Reasoning Memory improves diverse open-weight reasoning models on math, science, and coding tasks, scales well with both datastore size and inference budget, and remains robust across several implementation choices. More broadly, our results suggest a promising direction for both the reasoning and RAG communities: building systems that can accumulate, retrieve, and reuse procedural knowledge over time.

7.6 Limitations and Reproducibility

Data and Reproducibility All datasets used in this work are existing public resources and are used as-is for evaluation and corpus construction. We do not redistribute any model outputs, datasets, or source corpora. Our experiment code and scripts for reproducing the corpus construction pipeline from the original sources are publicly available at `https://github.com/facebookresearch/reasoning-memory`. Corpus statistics are reported in Section 7.2.1. This work does not involve human annotators or human subjects.

Limitations Our datastore is built from reasoning trajectories that span the evaluated math, science, and coding domains. Although Nemotron applies an explicit test-contamination removal process [NGM25], this alignment does not eliminate benchmark-overfitting risk. Our domain-specific and OpenThoughts3 results provide evidence of cross-domain and cross-source transfer, but do not establish generalization to arbitrary out-of-domain procedural corpora. We also do not measure what fraction of the 32M entries drives the gains or evaluate reasoning models below 7B parameters.

CHAPTER 8

Evolving Organizational Reflection Skills for Digital Agents

Long-horizon knowledge work is becoming an important frontier for large language model (LLM) digital agents, with direct relevance to economically valuable professional tasks. Producing a report, spreadsheet, analysis, or recommendation often requires sustained reasoning over heterogeneous information, repeated tool use, and the creation and refinement of complex artifacts. Recent benchmarks such as GDPval, JobBench, APEX-Agents, and Harvey's Legal Agent Benchmark (LAB) increasingly evaluate agents in precisely these settings [PDP25, LFX26, VMF26, GPP26]. Together with increasingly capable language models and general-purpose coding-agent harnesses, these advances make it possible for a single agent to carry out substantial portions of professional knowledge work. Yet completing the requested actions is not always enough: the resulting artifact must also be ready for others to review, consume, and act on.

A central challenge is that professional readiness is rarely determined from a single perspective. The same artifact may need to satisfy the expectations of stakeholders and customers, each of whom may notice different shortcomings. Human workers navigate such demands often by operating within organizations that expose their work to colleagues across different roles and functions [AT91, Car02]. Autonomous agents, in contrast, are typically left with two choices: wait for a human user to provide additional guidance, or attempt to identify and correct their own shortcomings through self-reflection. To bridge this gap, we ask:

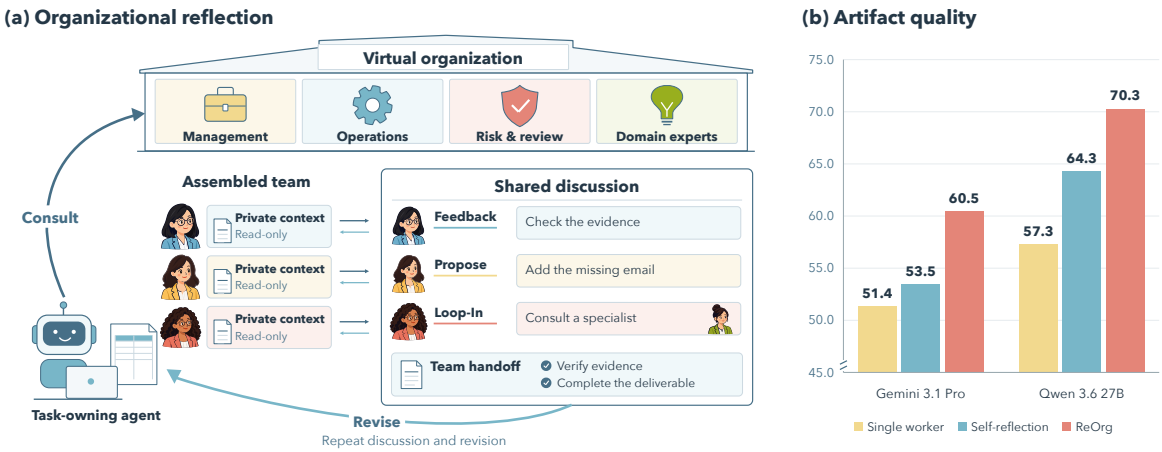


Figure 8.1: **Organizational reflection around a single worker.** (a) ReOrg consults a virtual organization of simulated stakeholders with separate read-only contexts. An assembled team produces a handoff that guides repeated revisions by the same task-owning worker. (b) GDPval win+tie rates compare a single worker, one round of self-reflection, and ReOrg with up to three reflection rounds.

Can we situate an agent with an organization where stakeholders serve as surrogate users?

Our pilot study on GDPval provides initial evidence for this direction. Self-reflection improves Gemini 3.1 Pro from 43.0% to 48.7% win+tie, while fixed role reflection with a manager reaches 53.2% and feedback from three stakeholder roles reaches 55.7% (Section 8.5.1). Moreover, repeated organizational feedback continues to improve performance across Gemini 3.1 Pro [Goo26a], Gemini 3.7 Flash [Goo26b], and Qwen3.6-27B [Qwe26b].

Motivated by this idea, we introduce **ReOrg**, which places a single long-horizon worker inside a virtual organization dedicated to reflecting on and improving its work (Section 8.2). As shown in Figure 8.1, the worker remains solely responsible for revising the task artifact, while an *Organizational Reflection Layer* provides the surrounding organizational structure. When reflection is needed, a planner assembles simulated stakeholders that independently inspect the current work and contribute from different organizational perspectives. Stakeholders operate through three communication actions—*Feedback*, *Propose*, and *Loop-In*—while a directed communication structure determines how contexts are accumulated

and propagated between stakeholders. Their interaction produces a compact handoff that guides the worker's next revision.

Furthermore, **ReOrg** posits that the underlying organizational patterns may recur and be reused across different tasks and work states: whose perspectives to involve, how they should communicate, and how their findings should be propagated. Inspired by organizational-routine theory [FP03], we represent these recurring patterns as reusable *organizational reflection skills*. ReOrg learns a library of such skills from a small stream of unlabeled tasks using evolutionary beam search and an unsupervised model-based judge, while jointly learning a planner that decides when to stop reflecting and which skill to retrieve when further reflection is useful (Section 8.3). This allows ReOrg to accumulate organizational knowledge about how to obtain and coordinate feedback, complementing procedural knowledge for task execution [WMF25, FLW26, OYH25].

We evaluate ReOrg on GDPval, JobBench, and LAB against procedural knowledge, self-reflection, and organizational reflection baselines. After learning from only 20 unlabeled GDPval tasks, up to three reflection rounds raise GDPval win + tie rate from 51.4% to 60.5% for Gemini 3.1 Pro and from 57.3% to 70.3% for Qwen 3.6 27B. The GDPval-trained libraries improve JobBench and LAB without target-domain adaptation, and Qwen shows gains over the single worker and random organization in all six cross-domain transfers (Section 8.5.2, Section 8.5.3). Skills learned with Gemini and Qwen also transfer to GPT 5.6 Luna, raising GDPval win + tie rate from 72.4% to 75.1% and 76.2%, respectively (Section 8.5.4). Accuracy–cost analysis shows that ReOrg reaches higher performance with additional inference tokens (Section 8.5.5). Together, these results support organizational knowledge as a transferable complement to procedural knowledge, enabling agents to learn how to seek, structure, and reuse feedback that improves their work.

8.1 Related Work

Long-horizon knowledge work Long-horizon knowledge work requires more than executing a sequence of actions: the resulting work must satisfy professional acceptance criteria. OfficeBench [WCZ24] and TheAgentCompany [XSL25] benchmark cross-application workplace activity, while GDPval [PDP25], JobBench [LFX26], APEX-Agents [VMF26] and Harvey's Legal Agent Benchmark (LAB, Grupen et al. [GPP26]) evaluate professional tasks using expert judgments or rubrics. Acceptance can require factual correctness, domain-specific analysis, presentation, and downstream usability, which need not be exhaustively stated in the request. For example, LAB pairs brief partner instructions with detailed criteria reflecting scrutiny by partners and clients. These benchmark designs mirror a broader property of organizational knowledge work: artifact quality is often evaluated from multiple stakeholder perspectives, with relevant knowledge distributed across roles and functions [AT91, Car02]. Inspired by this view, we consider organizational reflection as a natural complement to task execution, helping a worker anticipate the expectations of those who will review, consume, or act on its output.

Self-reflection and external feedback Once a worker has produced an initial artifact, an important question is how to identify what should be improved next. Prior work has explored this through self-reflection and external feedback. Self-Refine [MTG23] alternates self-feedback and revision. Reflexion [SCG23] carries verbal reflections across attempts. CRITIC [GSG24a] grounds critique in tool-based validation. Other work introduces additional perspectives: MetaGPT [HZC24] incorporates review into role-based collaboration, ARCANE [MGA25] translates stakeholder preferences into guidance for a worker, and MAJ-EVAL [CLW26] uses stakeholder personas to produce multidimensional evaluations. ReOrg builds on this broader view of feedback, but keeps task execution centralized in a single worker: stakeholder agents inspect and discuss the evolving artifact, while only the worker is responsible for revising it.

Agent Continual Learning and Skills Beyond improving a single artifact, agents can also accumulate reusable knowledge from experience across tasks. Existing work largely focuses on knowledge that helps the agent execute tasks more effectively: Voyager [WXJ24] stores executable behaviors, Agent Workflow Memory [WMF25] induces recurring workflows, while MemP [FLW26] and ReasoningBank [OYH25] distill procedural instructions and reasoning strategies from prior trajectories. More recent systems such as SkillOS [OYC26] and SkillNet [LZX26] study how such skills can be curated, evaluated, and maintained over time. Inspired by organizational-routine theory [FP03], we consider a complementary form of reusable knowledge: recurring patterns for how an organization reflects on and improves work. ReOrg therefore learns organizational knowledge about who should contribute, how feedback should be coordinated, and when reflection is useful, complementing procedural knowledge for task execution.

8.2 Organizational Reflection Layer

ReOrg places a knowledge-work agent within a virtual organization of simulated stakeholders that reflect on its evolving work. The worker retains ownership of the deliverable throughout the process. The reflection layer selects which perspectives to consult, structures how observations are exchanged, and returns guidance for the worker's next revision. Its reusable unit is an *organizational reflection skill*, which specifies a concrete reflection process.

8.2.1 Problem Formulation

Let $x = (u, \mathcal{R})$ denote a task packet with request u and reference materials $\mathcal{R}$. A tool-using worker W produces an initial artifact A_0, while a work chronicle H_r records progress and prior advice. At reflection round r, a planner either stops or selects a reflection skill s_r from the library $\mathcal{L}$. The selected skill produces stakeholder contributions F_r, which guide the

worker's next revision:

$$F_r = \text{Reflect}(x, A_{r-1}, H_{r-1}; s_r), \qquad A_r = W(x, A_{r-1}, H_{r-1}, F_r). \tag{8.1}$$

The process stops when the planner judges the current work ready or the reflection budget is exhausted.

8.2.2 Inference Pipeline

ReOrg alternates between task execution and organizational reflection, allowing the worker to improve its artifact without waiting for additional user feedback. The worker first produces A_0 using its ordinary tools. At each reflection round r, the planner either stops or retrieves a skill $s_r \in \mathcal{L}$ based on the current task and work state. The selected skill executes $\text{Reflect}(x, A_{r-1}, H_{r-1}; s_r)$ to produce stakeholder contributions F_r, which the worker uses to revise the artifact into A_r. This process repeats until the planner stops or the reflection budget is exhausted. During evaluation, the skill library and retrieval knowledge remain fixed.

To implement Reflect, we execute the stakeholder process specified by s_r in separate read-only contexts around the worker. Each stakeholder receives the task, current artifact, work chronicle, and any upstream contributions routed by the skill. Stakeholders can inspect files and verify artifact contents with local tools, but cannot modify the deliverable. Their contributions propagate according to the skill structure and are collected into F_r for the worker's next revision. This allows multiple organizational perspectives to inform reflection while preserving a single worker as the work owner.

8.2.3 Representing Organizational Reflection Skills

To reuse reflection across tasks, we separate the task-specific content of feedback from the organizational process that produces it. We represent a one-round reflection skill as a directed acyclic graph $s = (V, E)$. Each node $v \in V$ combines a stakeholder role ρ_v with a

communication schema a_v, while each edge $(v_i, v_j) \in E$ routes the contribution and context of v_i to v_j. The graph therefore specifies which perspectives participate and how their observations are combined, while concrete feedback is generated from the current task and artifact.

We instantiate the virtual organization through a catalog of stakeholder roles defined relative to the task owner. Our experiments use roughly ten roles, including a direct manager, upstream requester, same-job teammate, legal and risk reviewer, and cross-functional partner. We pair these roles with three communication schemas:

- **Feedback** identifies concrete defects, risks, and improvements.
- **Propose** suggests changes that better satisfy stakeholder needs.
- **Loop-In** consults another stakeholder and returns a consolidated discussion handoff.

Roles determine whose perspective is elicited, while communication schemas determine how that perspective contributes to reflection. Because a skill stores only these roles, schemas, and their communication structure, rather than task-specific feedback, it represents a task-agnostic organizational routine that can be instantiated on new tasks and artifacts. This makes reflection skills plug-and-play across domains and provides a reusable unit for continually learning organizational knowledge.

8.3 Continually Learning Organizational Reflection Skills

ReOrg continually learns organizational knowledge from a stream of unlabeled tasks by evolving a reflection-skill library $\mathcal{L}$ and accumulating planner knowledge M. Figure 8.2 and Algorithm 8.1 illustrate this process.

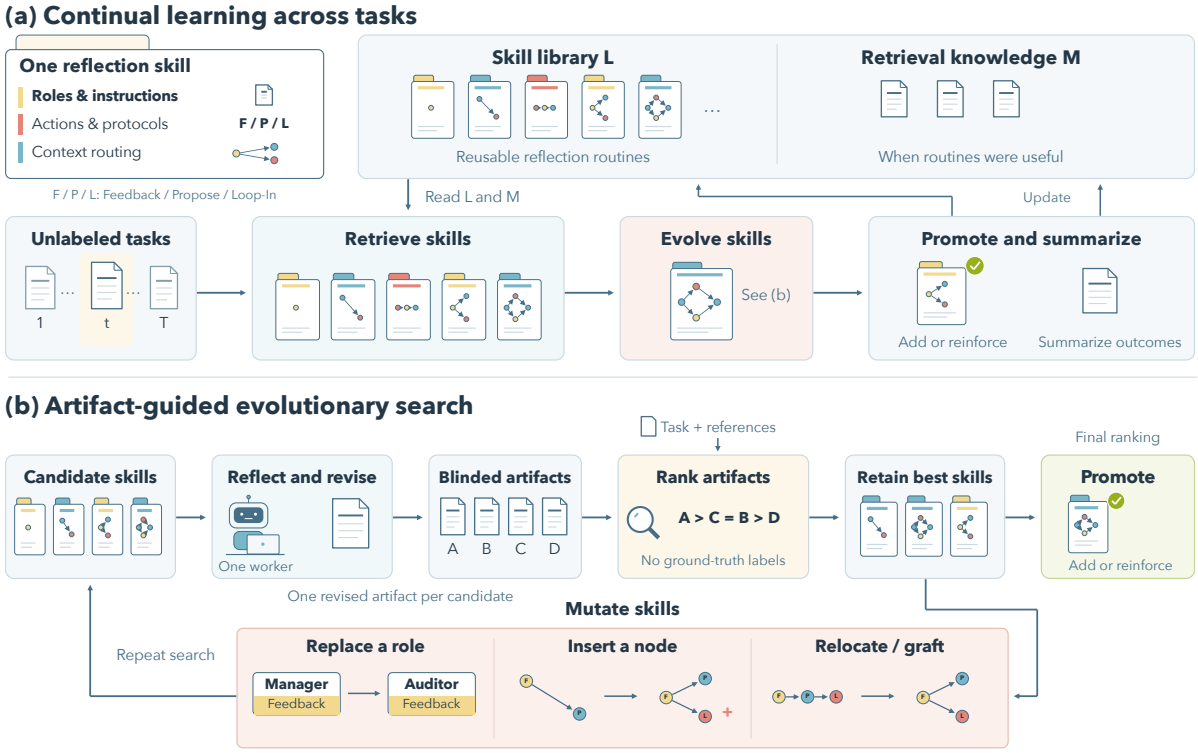


Figure 8.2: **Continual learning of reflection skills and retrieval knowledge.** (a) For each unlabeled task, ReOrg retrieves and evolves skills, then promotes useful structures and summarizes outcomes into retrieval knowledge for subsequent tasks. (b) A label-free judge compares blinded artifacts produced after reflection and worker revision. The resulting ranks guide skill selection, further mutation, and final promotion.

8.3.1 Artifact-Guided Skill Evolution

For each adaptation task x_t, the planner retrieves skills from $\mathcal{L}_{t-1}$ using its accumulated knowledge M_{t-1}. These skills seed an evolutionary beam search over stakeholder roles, communication schemas, and their interaction structure. Each candidate guides a worker revision and is evaluated through the *resulting artifact*.

We use the same LLM as a label-free judge to rank candidate artifacts by task fulfillment. The judge receives the task materials and blinded candidate submissions, which it inspects with local tools without privileged information. These relative judgments guide beam selection and skill promotion.

Algorithm 8.1 Continual learning of organizational reflection skills

Require: adaptation tasks $x_{1:T}$, initial library $\mathcal{L}_0$, planner knowledge M_0

for $t = 1, \ldots, T$ **do**
  $B \leftarrow \text{Retrieve}(x_t, \mathcal{L}_{t-1}, M_{t-1})$
  Execute skills in B and rank their resulting artifacts
  for each search step **do**
    $C \leftarrow \text{Mutate}(B)$
    Execute C and rank artifacts produced by $B \cup C$
    $B \leftarrow$ highest-ranked skills
  end for
  $\mathcal{L}_t \leftarrow \text{Promote}(\mathcal{L}_{t-1}, B)$
  $M_t \leftarrow M_{t-1} \cup \text{Summarize}(x_t, \text{search outcomes})$
end for
return frozen $(\mathcal{L}_T, M_T)$

8.3.2 Library and Planner Co-Evolution

After each task, we add strong new skills to $\mathcal{L}_t$ and reinforce rediscovered skills without creating duplicates. We also summarize the search outcomes into retrieval insights describing which organizational patterns appeared useful or unhelpful. These insights accumulate in a Markdown document M_t that guides skill selection on subsequent tasks.

After learning the one-round library, we collect state-conditioned precedents by comparing additional reflection skills from intermediate work states, including the option of leaving the artifact unchanged. These precedents are summarized into planner knowledge for later-round selection while the skill library remains fixed. At evaluation, both $\mathcal{L}$ and M are frozen. The completion gate in Section 8.2.2 determines whether another round is needed, and the planner uses the precedents to select an unused skill.

8.3.3 Search Configuration

Adaptation proceeds sequentially, with each task using the preceding library and retrieval lessons. Candidate executions and evaluations can run in parallel within a task. The five seeds are an empty graph, manager Feedback, manager Propose, manager Loop-In, and manager Feedback followed by Propose.

We retrieve five candidates per task and use beam width three, up to three distinct mutations per parent, and three search iterations. This permits at most $5 + 3 \times 3 \times 3 = 32$ candidate executions before deduplication and failures, with at most ten stakeholder nodes per reflection round. Mutations insert Feedback, Propose, or Loop-In nodes, relocate nodes, or replace roles; insertions can be sequential or parallel. Online comparisons use four blinded artifacts, with at least three appearances per candidate. Promotion compares five distinct candidates where possible and retains the top three.

Artifact comparison. The online judge inspects task-facing materials and anonymously labeled artifacts with local tools, without benchmark answers, hidden rubrics, candidate graphs, or role assignments. It returns ordered tiers, hard-invalid labels, and a brief rationale after inspecting every submission. Validation requires each candidate label to appear exactly once and invalid labels to belong to the presented set. Comparison groups balance exposure and include required parent–child comparisons. Execution failures and invalid judgments are recorded separately, and promotion requires a sufficient valid pool.

8.3.4 Later-Round Planning and Frozen Deployment

Learning later-round retrieval. With the one-round library frozen, we revisit each adaptation task and execute its first retrieved skill. A state projector summarizes the task, attributed suggestions, and completed work from the task and chronicle. We sample eight unused, nonempty skills without replacement, distributing slots across populated node-count levels in a seeded order and revisiting levels as needed. Each branch performs reflection and worker revision. The ranker compares these artifacts with the unchanged parent as a STOP candidate. A separate prompt distills the comparisons into state-conditioned retrieval lessons without changing the skill library.

Deployment. First-round retrieval ranks five skills and executes the first. Before a later round, a completion gate scores the projected work state from 1 to 10. Gemini and Qwen stop at a score of at least 9. Otherwise, the retriever ranks nine available nonempty skills and `leave_unchanged` using the current state and learned precedents. The controller selects the highest-ranked non-stop skill, even when `leave_unchanged` ranks first, and excludes previously used identities and execution fingerprints. Thus the completion gate controls stopping, while retrieval selects the next skill. An empty skill still invokes worker revision. Evaluation pairs library and lesson snapshots from the same task boundary and keeps both frozen. Benchmark adapters preserve the learned graph while staging task-specific inputs and submissions.

8.4 Evaluation Setup

8.4.1 Benchmarks and Evaluation Protocol

We evaluate artifact-producing knowledge work on GDPval, JobBench, and Harvey's Legal Agent Benchmark (LAB), covering diverse professional domains and acceptance criteria. Workers and stakeholders receive only the task-facing inputs. Benchmark reference outputs and hidden rubrics are reserved for final evaluation and are not used by the reflection layer or its online learning judge.

Tasks and metrics. **GDPval** [PDP25] evaluates professional deliverables against expert-produced reference work. Our operational cohort contains 185 tasks. We compare each submitted artifact with its expert reference using a panel of Gemini 3.1 Pro, Qwen 3.6, and Claude Sonnet 4.6 judges and aggregate their categorical preferences by majority. The reported metric is win + tie rate. Sections 8.4.3 and 8.4.4 describe the reference-based judge and our meta-evaluation results.

JobBench [LFX26] evaluates work against task-specific rubrics. We use 128 tasks: 65

from its main subset and 63 from its easy subset. The native evaluator, configured with Gemini 3.5 Flash High, produces a rubric-weighted normalized score for each task. We average these scores over tasks.

LAB [GPP26] evaluates legal artifacts against detailed professional criteria. We use the fixed 200-task native test subset and the native criterion-level evaluator configured with Gemini 3.1 Pro. The main results report rubric-item pass rates.

Adaptation and transfer. For the main ReOrg configuration, we process 20 unlabeled GDPval tasks and freeze the resulting library and planner knowledge before evaluation. The cross-domain study also learns libraries from JobBench and LAB, using 20 adaptation tasks per domain.

8.4.2 Agents, Baselines, and Budgets

Agents and execution. Our main comparison evaluates Gemini 3.1 Pro and Qwen 3.6 27B. GPT 5.6 Luna is the target of a separate cross-model transfer study, and Gemini 3.7 Flash is additionally included in the pilot. Gemini and Qwen use a Stirrup-based harness, while Luna uses its native Codex harness. In the main comparison, the worker and all reflection and learning components use the same backbone. In the cross-model study, Luna executes reflection skills learned with Gemini or Qwen.

Baselines. We compare with a single worker and three groups of baselines. *Procedural knowledge* methods include adaptations of ReasoningBank [OYH25], AWM [WMF25], MemP [FLW26], and ReasoningBank + MaTTS to artifact-producing tasks. These methods learn reusable guidance from unlabeled experiences and freeze their memory before evaluation. *Self-reflection* asks the worker to review and revise its own artifact without external feedback. *Organizational reflection* includes *fixed role reflection*, which uses direct-manager feedback, and *random organization*, which uses a randomly generated graph of

generic reviewers. Both are followed by one worker revision. Random organization controls for additional reflection effort without isolating topology from role prompts and action types.

Reflection and learning budgets. ReOrg uses a frozen skill library for up to one, two, or three reflection rounds, with later-round planning described in Section 8.3.4. ReasoningBank + MaTTS uses 32 training rollouts per adaptation task, matching ReOrg's nominal maximum number of candidate executions, followed by a single memory-augmented solve at evaluation. Random organization uses fixed node budgets. Section 8.3.3 gives the search configuration.

Harness and baseline implementation. Stakeholders inspect artifacts with local tools and have no web retrieval. GDPval workers may retrieve task-relevant web evidence, JobBench permits guarded retrieval for main tasks but disables it for easy tasks, and LAB disables web retrieval. Luna uses its native Codex harness with extra-high reasoning effort and permits one level of native subagent delegation within worker calls. Ambient skills, memories, plugins, extra tools, and inherited project instructions are disabled.

ReasoningBank, MemP, and offline AWM share 20 initial-worker experiences per benchmark and backbone. Their induction excludes expert outputs, hidden rubrics, gold answers, and final-judge results. At evaluation, BM25 retrieves three complete entries from each frozen library. Random organization uses five generic reviewer nodes for Gemini and four for Qwen. ReasoningBank + MaTTS uses 32 fresh solves per adaptation task, followed by joint memory induction, and transfers its GDPval library unchanged to JobBench and LAB. This matches ReOrg's nominal candidate-execution count, but not its token cost.

8.4.3 Evaluation Details

8.4.3.1 Task Cohorts and Evaluation Inputs

Workers, stakeholders, and online ranking receive only task-facing materials. Final evaluation adds hidden rubrics and, for GDPval, expert artifacts.

GDPval's 185-task cohort includes 20 adaptation and 165 held-out tasks. JobBench comprises 65 main and 63 easy tasks, with 108 held out after its 20-task in-domain adaptation. All 128 JobBench tasks are unseen during GDPval-only adaptation. LAB uses 200 native test tasks disjoint from its adaptation pools. Main GDPval and in-domain JobBench results include adaptation tasks. Each task is counted once, with incomplete evaluations recorded separately.

8.4.3.2 GDPval Artifact Evaluation

A fresh workspace contains the request, supplied references, signed-point rubric, and candidate and expert artifacts under randomized Response A/B labels with file manifests. Gemini 3.1 Pro, Qwen 3.6 27B, and Claude Sonnet 4.6 independently inspect them using code execution and image viewing, without web access. Each returns a rationale, integer raw scores, and a consistent categorical preference. Negative criteria impose penalties, and equal scores yield a tie. After restoring candidate/expert identities, the controller takes the categorical majority, treating a win/tie/loss split as a tie. Missing or invalid panel members make the evaluation incomplete.

GDPval performance is

$$\mathrm{WinTie} = \frac{N_{\mathrm{win}} + N_{\mathrm{tie}}}{N_{\mathrm{cohort}}}. \tag{8.2}$$

We retain win, tie, loss, and unavailable counts. Each model judges a submission pair once. Section 8.4.4 reports its system-level calibration.

8.4.3.3 JobBench and LAB Scoring

JobBench's native evaluator uses Gemini 3.5 Flash at high thinking, extracted submission contents, and images when criteria require visual inspection. A rubric item earns its weight only when all criteria pass. Earned weight divided by total weight gives the task score, recorded to four decimal places. The benchmark score averages these normalized task scores:

$$S_{\text{JobBench}} = \frac{65\, S_{\text{main}} + 63\, S_{\text{easy}}}{128}. \tag{8.3}$$

Empty submissions receive the native failure score, while infrastructure failures remain unavailable.

LAB's native Gemini 3.1 Pro evaluator returns a binary verdict and rationale for each criterion using its matching instructions, task description, and relevant deliverables. We record passed criteria p_t, total criteria n_t, submission eligibility, and whether all criteria pass. Historical reports use two aggregate rates:

$$S_{\text{macro}} = \frac{1}{T}\sum_{t=1}^{T}\frac{p_t}{n_t}, \qquad S_{\text{pooled}} = \frac{\sum_{t=1}^{T} p_t}{\sum_{t=1}^{T} n_t}. \tag{8.4}$$

The macro rate weights tasks equally, while the pooled rate weights individual criteria equally. Explicitly incomplete submissions receive zero passed criteria and are marked ineligible. Missing run records remain unavailable.

8.4.4 GDPval Judge Meta-Evaluation

We assess system ordering using the June 25 judge-development study, which evaluated nine systems with five candidate judges and one run per system–judge pair. The systems are Gemini 3.5 Flash, Gemini 3.1 Pro, Gemma 4 26B A4B, GPT-OSS-120B, Qwen 3.6 27B, GPT 5.4 mini, Claude Opus 4.8, Claude Sonnet 4.6, and Claude Haiku 4.5. The comparison target is their ordering in the recorded public GDPval-AA Elo roster.

Matching successful primary judgments with explicit candidate/expert identities across the three panel members yields 1,608 system–task cells, or 170–185 tasks per system. For

Judge or aggregation	Kendall τ_b	Pair order (>150 Elo)	LOO RMSE
Gemini 3.1 Pro	0.9	31/31	85.0
Qwen 3.6 27B	0.9	31/31	87.7
Claude Sonnet 4.6	1.0	31/31	76.5
Three-judge hard majority	1.0	31/31	81.0

Table 8.1: System-level judge calibration against the recorded GDPval-AA ordering on 1,608 matched system–task cells. RMSE is measured in Elo units.

calibration, wins, ties, and losses receive 1, 1/2, and 0, averaged within each system. We report Kendall's τ_b against Elo ordering and ordering accuracy for pairs separated by more than 150 Elo points. Leave-one-system-out RMSE fits an affine preference-to-Elo mapping on eight systems and predicts the ninth. Additional A/B swap audits are excluded.

The judges agree unanimously on 1,209 cells (75.2%), form a two-of-three majority on 365 (22.7%), and return three different labels on 34 (2.1%). A strict majority exists for 97.9% of cells, with pairwise agreement of 80.7–84.1%. The panel recovers the recorded system order, while Sonnet has the lowest RMSE. Because this roster also informed judge selection, the calibration is exploratory.

8.5 Experiments and Analysis

We evaluate stakeholder reflection, learned reflection skills, and transfer across domains and models. We then analyze inference cost and the structure of the learned libraries.

8.5.1 Pilot Study: Reflection with Stakeholder Perspectives

We begin with a question: *Can simulated stakeholder perspectives help a worker improve its artifact beyond self-reflection?* We hypothesize that stakeholders bring different expectations to the same deliverable and can identify shortcomings that the worker's own review may miss. We examine this idea in a GDPval pilot comparing self-reflection with stakeholder-guided revision.

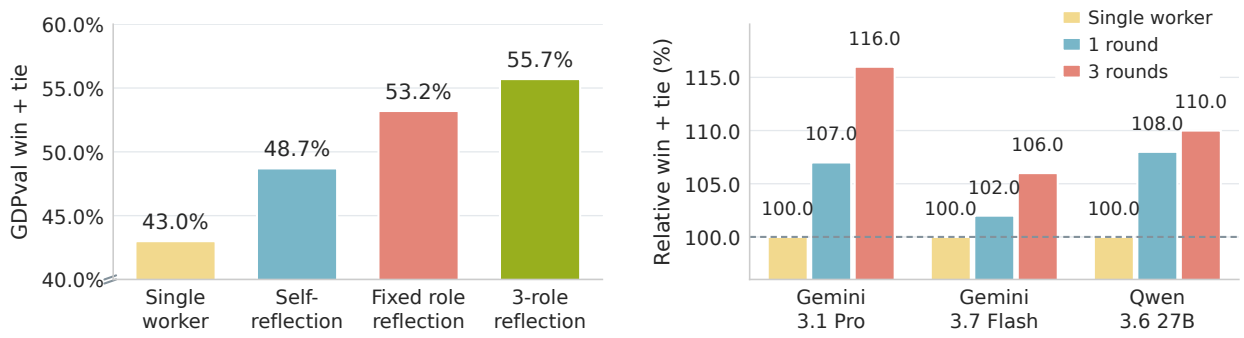


Figure 8.3: Pilot study on GDPval. (a) Win + tie rates for Gemini 3.1 Pro under four reflection conditions, shown on a 40.0–60.0% scale. (b) Win + tie rates with one or three organizational reflection rounds, normalized by each model's single-worker rate (100.0%).

Figure 8.3(a) shows that, for Gemini 3.1 Pro, self-reflection raises GDPval win + tie rate from 43.0% to 48.7%, while fixed role reflection with a manager reaches 53.2%. The 4.5 percentage-point gain over self-reflection suggests that an external stakeholder perspective provides useful guidance beyond the worker's own critique. Reflection involving three stakeholder roles across three revision rounds further raises performance to 55.7%.

We next ask whether organizational reflection remains useful after the first revision. In Figure 8.3(b), increasing the budget from one to three rounds improves performance across all three backbones. Expressed as a percentage of each model's single-worker rate, performance rises from 107.0% to 116.0% for Gemini 3.1 Pro, from 102.0% to 106.0% for Gemini 3.7 Flash, and from 108.0% to 110.0% for Qwen 3.6 27B. Together, these observations motivate learning reusable reflection processes that can guide successive revisions of an evolving artifact.

8.5.2 Main Results: ReOrg and Strong Baselines

Learned organizational reflection improves artifact quality. Table 8.2 shows that ReOrg improves GDPval performance after learning reflection skills from only 20 unlabeled tasks. A single reflection round reaches 60.0% for Gemini and 68.7% for Qwen, compared with 51.4% and 57.3% for their single-worker baselines. These correspond to gains of 8.6

Method	Gemini 3.1 Pro			Qwen 3.6 27B		
	GDPval	JobBench	LAB	GDPval	JobBench	LAB
Single worker	51.4	28.7	50.9	57.3	45.5	68.7
ReasoningBank	47.0	28.6	56.7	57.3	43.3	72.7
ReasoningBank + MaTTS	50.3	30.5	58.6	60.0	46.2	73.0
AWM	48.1	30.6	57.2	60.5	43.7	72.1
MemP	47.6	31.2	57.5	59.5	43.6	76.0
Self-reflection (1 round)	53.5	29.8	59.3	64.3	45.5	74.9
Fixed role reflection	52.4	30.0	59.6	61.6	46.5	76.0
Random organization	53.5	30.0	62.0	64.3	45.7	75.2
ReOrg (1 round)	60.0	30.6	61.9	68.7	46.2	76.6
ReOrg (2 rounds)	60.5	30.8	62.0	69.2	47.7	76.9
ReOrg (3 rounds)	60.5	31.0	62.0	70.3	47.7	76.9

Table 8.2: Main evaluation results (%). ReOrg and ReasoningBank + MaTTS learn from 20 unlabeled GDPval tasks and transfer their learned resources to JobBench and LAB. ReasoningBank, AWM, and MemP adapt in-domain on 20 tasks per benchmark. Row colors distinguish baseline families (procedural knowledge, self-reflection, and organizational reflection), with separate colors for the single worker and ReOrg.

and 11.4 percentage points. Both models also outperform self-reflection, fixed role reflection, random organization, and the procedural knowledge baselines on GDPval.

Further reflection builds on the first-round gains. Allowing up to three reflection rounds raises GDPval performance to 60.5% for Gemini and 70.3% for Qwen. The first round provides most of the improvement, while subsequent rounds add 0.5 and 1.7 percentage points, respectively. All three settings reuse the same frozen skill library, with later rounds selecting skills based on the evolving work state. The round budgets are upper limits, and tasks retain their current artifact when the planner decides to stop.

Reflection skills transfer across professional domains. The GDPval-trained library also improves performance on JobBench and LAB without adaptation to either target domain. On JobBench, three rounds of ReOrg reach 31.0% for Gemini and 47.7% for Qwen.

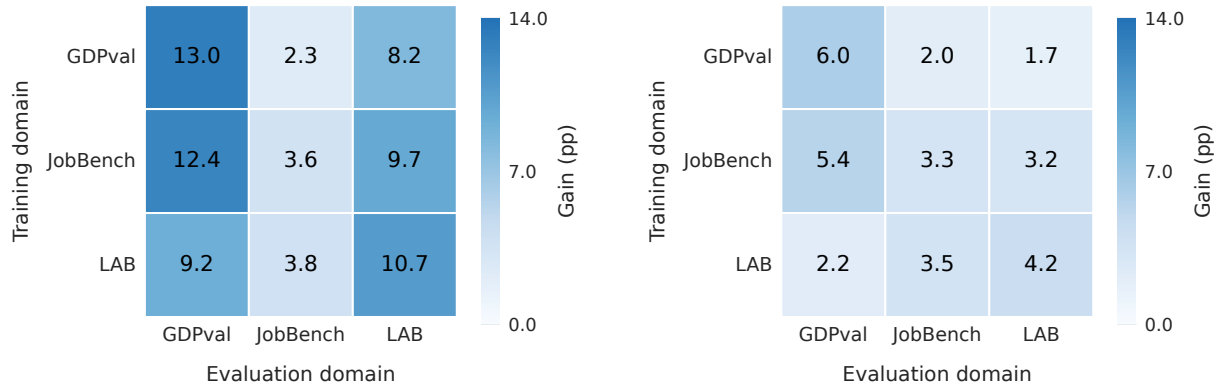


Figure 8.4: Cross-domain transfer with Qwen 3.6 27B, measured in percentage-point gains over the single worker (left) and random organization (right). Each library is learned from 20 unlabeled tasks in the row domain and evaluated in the column domain.

Qwen outperforms all listed baselines, while Gemini approaches the 31.2% achieved by in-domain MemP. The LAB results likewise improve over the single worker, with smaller differences among the strongest reflection-based methods.

8.5.3 Cross-Domain Transfer of Reflection Skills

Using Qwen 3.6 27B, we build a library from 20 unlabeled tasks in each of GDPval, JobBench, and LAB, then evaluate each frozen library across all three domains. Figure 8.4 reports gains over the single worker and random organization. All nine source–target combinations improve over both baselines, including all six cross-domain transfers. For these cross-domain settings, gains over random organization range from 1.7 to 5.4 percentage points. For example, JobBench-trained skills improve GDPval by 5.4 points and LAB by 3.2 points, while LAB-trained skills improve JobBench by 3.5 points. The most useful source domain is not always the target domain itself. A transferred skill retains its stakeholder roles and communication structure, while each stakeholder generates feedback from the new task and artifact.

Method / source library	GPT 5.6 Luna
Single worker	72.4
ReasoningBank + MaTTS	74.1
Self-reflection (1 round)	74.6
Random organization	75.2
ReOrg (Gemini 3.1 Pro skills)	75.1
ReOrg (Qwen 3.6 27B skills)	**76.2**

Table 8.3: Cross-model transfer to GPT 5.6 Luna on GDPval. Organizational reflection skills learned with Gemini 3.1 Pro or Qwen 3.6 27B are executed with Luna. All entries report GDPval win + tie rate (%). Row colors distinguish baseline families (procedural knowledge, self-reflection, and organizational reflection), with separate colors for the single worker and ReOrg.

8.5.4 Cross-Model Transfer

We next evaluate whether reflection skills learned with Gemini or Qwen remain useful when executed with GPT 5.6 Luna. Table 8.3 reports GDPval win + tie rates for the transferred libraries and Luna baselines. Gemini-trained skills reach 75.1%, while Qwen-trained skills reach 76.2%, compared with 72.4% for the Luna single worker. These are gains of 2.7 and 3.8 percentage points, respectively. Qwen-trained skills exceed the strongest listed baseline, random organization at 75.2%, by 1.1 points. Gemini-trained skills perform similarly to that baseline. Both transferred libraries also exceed self-reflection at 74.6% and ReasoningBank + MaTTS at 74.1%. These results support transfer to Luna, though the study does not evaluate transfer in every direction.

8.5.5 Further Analyses

Accuracy and inference cost. Figure 8.5 compares GDPval win + tie rate against mean token cost per evaluation task for the single worker, self-reflection, random organization, ReasoningBank + MaTTS, and ReOrg. For Gemini, ReOrg reaches 60.0% at 1.5 million tokens per task and 60.5% at about 1.6 million. The second and third rounds have the same reported performance and nearly identical cost. For Qwen, ReOrg increases from 68.7% at

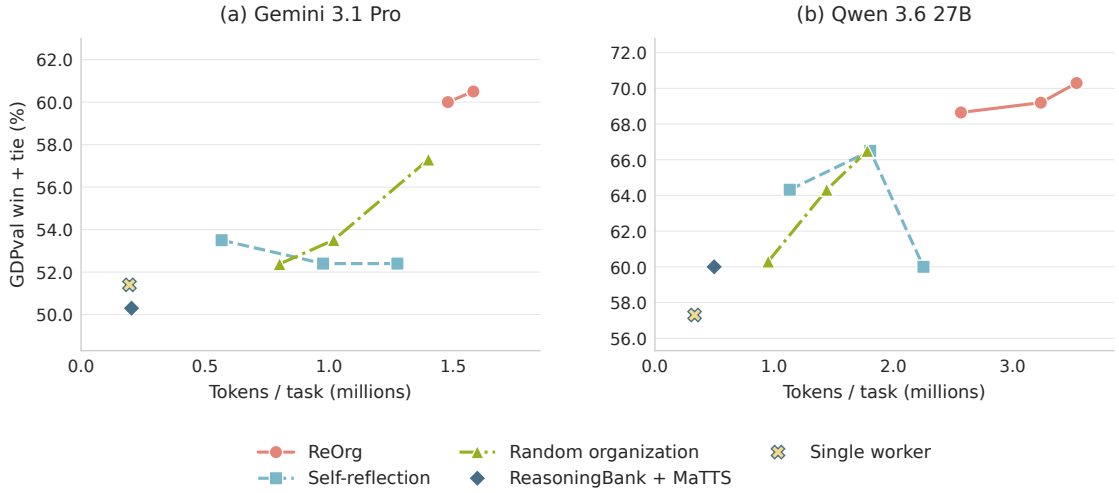


Figure 8.5: Accuracy–token-cost trade-offs on GDPval. Cost denotes mean input + output tokens per evaluation task, excluding skill construction and final judging. Lines connect configurations within each method family in increasing cost order.

2.6 million tokens to 70.3% at 3.5 million. The strongest baseline rates are 57.3% for Gemini and 66.5% for Qwen, achieved at lower cost than ReOrg. The higher performance therefore comes with additional inference cost, and the results do not establish an advantage at every token budget.

Library growth and learned structures. After 20 GDPval adaptation tasks, the five-seed library grows to 42 registered skills for Gemini and 40 for Qwen (Figure 8.6a). The five seeds remain available throughout adaptation, while repeated selection can reinforce an existing skill without adding a new record. Both final libraries cover all ten stakeholder roles and all three communication actions. Among their stored action nodes, Feedback accounts for 40.1% in Gemini and 50.6% in Qwen, while Loop-In accounts for 34.3% and 21.0%, respectively (Figure 8.6b).

The role distribution also differs across the two libraries (Figure 8.6c). Direct managers and upstream requesters appear in most skills in both libraries. Gemini more often includes experienced mentors, while Qwen more often includes same-job teammates, legal and risk reviewers, and cross-functional partners. Learning thus expands the available reflection

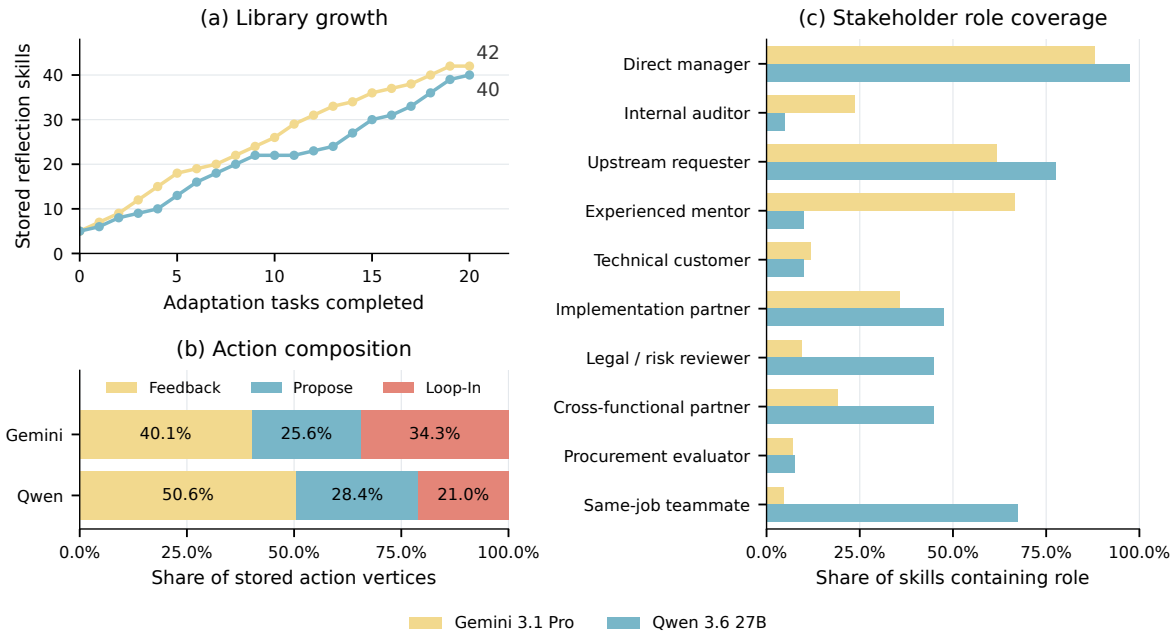


Figure 8.6: Growth, action composition, and stakeholder role coverage of the GDPval reflection libraries. (a) Registered skills after each adaptation task. (b) Shares of stored action nodes in the final libraries. (c) Shares of final skills containing each role, counted once per skill. All panels include the five seed skills. Role counts exclude dynamically selected Loop-In partners.

processes while retaining distinct mixtures of stakeholder perspectives and communication actions. These frequencies describe the stored libraries, rather than the contribution of individual roles or actions to performance.

8.5.6 Case Study: Reviewing Municipal Well Data

A Qwen GDPval task requests a workbook covering wells in eight Illinois municipal systems and an email recommending candidates to a manager. Candidates must be active, 160–200 feet deep, and in sand-and-gravel aquifers. The initial workbook lists 33 wells and seven candidates across two worksheets but omits the email. Table 8.4 traces the revisions and candidate-versus-expert judge votes.

Artifact	Reflection before revision	Trio votes
Initial	Worker solve	Loss, 0:3
Round 1	Teammate Propose, manager Feedback, requester Loop-In, legal/risk Feedback, and cross-functional Feedback in sequence	Loss, 1:2
Round 2	The same five perspectives, with cross-functional Feedback moved ahead of the manager and requester	Win, 2:1
Round 3	Manager Feedback followed by manager Propose	Win, 3:0

Table 8.4: Reflection sequence and candidate-versus-expert votes for the municipal well task.

Round 1: completeness and decision risks. The teammate and manager flag the missing email, and the manager notes that the workbook lacks its claimed conditional formatting. Other feedback distinguishes five producing wells from two with zero recorded pumpage and raises municipal ownership, intergovernmental agreements, provenance, units, and the concentration of four candidates in Farmer City. The worker adds freeze panes, formatting, verification notes, and two candidate tiers, but still omits the email.

Round 2: spreadsheet usability. Moving cross-functional feedback after the teammate's proposal brings attention to plain-text URLs, ambiguous `None` values, and inconsistent worksheet formatting. The worker adds source hyperlinks, replaces ambiguous entries with `N/A`, applies shortlist conditional formatting, and carries unit and ownership caveats across both worksheets. The requester keeps zero-pumpage candidates in Tier 2 and recommends deferring distance analysis until a facility site is available. The email remains missing despite a majority win against the expert reference.

Round 3: completing the handoff. Manager Feedback and Propose target the missing email and an active-well check that could admit inactive wells. The worker adds an *Email to Manager* worksheet with all seven candidate IDs, depths, pumpage, tiered recommendations, verification and outreach steps, and key caveats. It also tightens screening to require official status `A`. The final workbook receives all three judge votes.

This selected case shows successive feedback addressing completeness, usability, and unresolved requirements. It does not isolate the effect of a particular role or skill.

8.6 Conclusion

We introduced ReOrg, which equips a single knowledge-work agent with a learned organizational reflection layer. Its reflection skills specify which stakeholder perspectives to consult, how they contribute, and how their observations are combined to guide revision. By learning these skills from unlabeled tasks, ReOrg accumulates reusable knowledge about how to obtain useful feedback while keeping the worker responsible for the deliverable. Experiments show substantial improvements on GDPval and useful transfer across professional domains. Skills learned with Gemini and Qwen also improve performance with GPT 5.6 Luna, supporting reuse across model backbones. The cost analysis highlights the additional inference required for the strongest results. These findings suggest that learning how to examine and improve work is a valuable complement to learning how to carry it out. Organizational knowledge thus provides a transferable resource for improving long-horizon agents.

CHAPTER 9

Conclusion and Future Directions

9.1 Conclusions

This dissertation studies how language models can operate in specific and evolving private worlds. We argue that reliable knowledge augmentation requires both models that recognize and navigate their knowledge boundaries and external knowledge layers that discover, structure, and evolve useful knowledge.

In Part I, we develop LongMemEval and LongMemEval-V2 to evaluate memory across conversations and agent experience. These benchmarks expose challenges in recovering relevant evidence, tracking changes, and applying knowledge from accumulated histories.

In Part II, we study selective knowledge access through Repoformer and Self-Routing RAG. These methods learn when to retrieve and which sources to consult, using external context more effectively while avoiding unnecessary retrieval.

In Part III, we study the construction of reusable knowledge through Deep Policy Research, Reasoning Memory, and ReOrg. These methods organize domain policies, reasoning procedures, and organizational reflection skills into resources that support later tasks. Together, they broaden the role of external knowledge to include guidance for solving problems and obtaining useful feedback.

Together, these contributions support systems that become familiar with their private worlds. Sustained deployment further requires deciding what to learn, how to retain it, and whether accumulated knowledge improves subsequent work.

9.2 Future Directions

Curiosity-driven exploration and interaction. A promising direction is to automate more of the forward-deployed engineering needed to adapt agents to unfamiliar workspaces. An agent could enter an organization as an intern, learn from its documents and interactions, and gradually develop expertise through continually maintained memory. ApprenticeBench [Neo26], which evaluates learning a job through onboarding and subsequent work, provides an early setting for studying this progression. Engram's study of a synthetic law firm also illustrates how prior exploration can build useful workspace knowledge [Eng26].

The central question is how an agent should choose what to learn when the value of that knowledge may emerge only in future tasks. Curiosity could guide the agent toward unfamiliar workflows, uncertain assumptions, and gaps in its understanding. Exploration would include reading records, inspecting tools, asking targeted questions, and learning from demonstrations or corrections. Such a system must balance completing the current task with investing in future competence, while accounting for the cost of exploration and human attention. It must also revise its memory as the workspace changes.

Self-calibrating context compression. A second direction concerns when external, non-parametric knowledge should be consolidated into model parameters. Repeatedly retrieving the same background information may be inefficient. Engram's law-firm study combines knowledge in model weights, text memory, and source documents [Eng26].

We envision a system that internalizes high-level domain intuition and frequently useful, stable facts, while preserving long-tail and rapidly changing details in external memory. This familiarity could help the model locate evidence and plan more accurate and efficient sequences of tool calls. External sources would supply precise details for the current task.

We refer to this direction as *self-calibrating context compression*. The system would learn both what knowledge merits consolidation and when its internal understanding is insufficient

or outdated. Reuse frequency, knowledge stability, and the relative costs of training and retrieval could guide these decisions. An important challenge is preserving the ability to recognize knowledge boundaries after consolidation, so that increasing familiarity does not lead the model to rely on stale assumptions. This would connect selective knowledge access with continual learning of the knowledge being accessed.

Extrinsic evaluation of memory. Finally, memory should be evaluated by how it benefits complete tasks over time. Intrinsic properties, such as whether stored information is correct and current, are important but do not fully determine its usefulness. Accurate knowledge may be difficult to retrieve, presented in an unusable form, or supplied too late to influence an action. Conversely, a compact memory may omit many details while retaining what is needed for successful work.

LongMemEval-V2 takes a step toward evaluating usability by measuring the answers produced by a fixed reader from retrieved context. Extending this evaluation to interactive tasks introduces a harder attribution problem: success and failure also depend on planning, tool use, and execution. Future benchmarks could compare agents with different memory systems under matched interaction budgets, and use controlled changes to memory access to identify where improvements arise. Repeated tasks in evolving workspaces would further reveal whether remembered experience reduces recurring errors and the need for human guidance. Evaluation should account for exploration, memory maintenance, and inference costs, so that improvements reflect the cumulative value of learning about a private world.

REFERENCES

[AJA24] Marah I Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat S. Behl, et al. "Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone." *CoRR*, **abs/2404.14219**, 2024.

[AKA25] Ammar Ahmed, Azal Ahmad Khan, Ayaan Ahmad, Sheng Di, Zirui Liu, and Ali Anwar. "Retrieval-of-Thought: Efficient Reasoning via Reusing Thoughts." *CoRR*, **abs/2509.21743**, 2025.

[AML24] Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, and Jian-Guang Lou. "Make Your LLM Fully Utilize the Context." *CoRR*, **abs/2404.16811**, 2024.

[Ant26] Anthropic. "Claude Opus 4.6 System Card." `https://www-cdn.anthropic.com/14e4fb01875d2a69f646fa5e574dea2b1c0ff7b5.pdf`, 2026. Accessed: 2026-04-23.

[AT91] S. J. Ashford and A. S. Tsui. "SELF-REGULATION FOR MANAGERIAL EFFECTIVENESS: THE ROLE OF ACTIVE FEEDBACK SEEKING." *Academy of Management Journal*, **34**(2):251–280, 06 1991.

[AWW24] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection." In *The Twelfth International Conference on Learning Representations*, 2024.

[BDP25] Islem Bouzenia, Premkumar T. Devanbu, and Michael Pradel. "RepairAgent: An Autonomous, LLM-Based Agent for Program Repair." In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pp. 2188–2200. IEEE, 2025.

[BJT22] Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. "Efficient training of language models to fill in the middle." *ArXiv preprint*, **abs/2207.14255**, 2022.

[BKK21] Sumithra Bhakthavatsalam, Daniel Khashabi, Tushar Khot, Bhavana Dalvi Mishra, Kyle Richardson, Ashish Sabharwal, Carissa Schoenick, Oyvind Tafjord, and Peter Clark. "Think you have Solved Direct-Answer Question Answering? Try ARC-DA, the Direct-Answer AI2 Reasoning Challenge." *CoRR*, **abs/2102.03315**, 2021.

[BKK22] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron

McKinnon, et al. "Constitutional AI: Harmlessness from AI Feedback." *arXiv preprint arXiv:2212.08073*, 2022.

[BMH22] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. "Improving Language Models by Retrieving from Trillions of Tokens." In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022.

[BPC20] Iz Beltagy, Matthew E. Peters, and Arman Cohan. "Longformer: The Long-Document Transformer." *arXiv:2004.05150*, 2020.

[BTG24] Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier de Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. "WorkArena++: Towards Compositional Planning and Reasoning-based Common Knowledge Work Tasks." In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[BTZ25] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. "LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks." In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 3639–3664. Association for Computational Linguistics, 2025.

[BWT18] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, I
textasciitilde nigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. "MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling." In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 5016–5026, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[Car02] Paul R. Carlile. "A Pragmatic View of Knowledge and Boundaries: Boundary Objects in New Product Development." *Organization Science*, **13**(4):442–455, August 2002.

[CKA25] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. "Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory." In Inês Lynce, Nello Murano, Mauro Vallati, Serena Villata, Federico Chesani, Michela Milano, Andrea Omicini, and Mehdi Dastani, editors, *ECAI 2025 - 28th European Conference on Artificial Intelligence, 25-30 October 2025, Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025)*, Frontiers in Artificial Intelligence and Applications, pp. 2993–3000. IOS Press, 2025.

[CLS24] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. "Teaching Large Language Models to Self-Debug." In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[CLW26] Jiaju Chen, Yuxuan Lu, Xiaojie Wang, Huimin Zeng, Jing Huang, Jiri Gesi, Ying Xu, Bingsheng Yao, and Dakuo Wang. "Multi-Agent-as-Judge: Aligning LLM-Agent-Based Automated Evaluation with Multi-Dimensional Human Evaluation." In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17389–17416, San Diego, California, United States, July 2026. Association for Computational Linguistics.

[Coz24] Coze. "Memory Overview Guide." `https://www.coze.com/docs/guides/memory_overview?_lang=en`, 2024. Accessed: September 15, 2024.

[CPW23] Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. "Walking Down the Memory Maze: Beyond Context Limit through Interactive Reading." *CoRR*, **abs/2310.05029**, 2023.

[CTJ21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. "Evaluating large language models trained on code." *ArXiv preprint*, **abs/2107.03374**, 2021.

[CWA23] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. "Adapting Language Models to Compress Contexts." In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 3829–3846. Association for Computational Linguistics, 2023.

[CWC23] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. "Dense X Retrieval: What Retrieval Granularity Should We Use?" *CoRR*, **abs/2312.06648**, 2023.

[CYD26] Weili Cao, Xunjian Yin, Bhuwan Dhingra, and Shuyan Zhou. "Coding Agents are Effective Long-Context Processors." *CoRR*, **abs/2603.20432**, 2026.

[DCE22] Bhuwan Dhingra, Jeremy R. Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W. Cohen. "Time-Aware Language Models as Temporal Knowledge Bases." *Transactions of the Association for Computational Linguistics*, **10**:257–273, 2022.

[DCX23] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. "Enhancing Chat Language Models by Scaling High-quality Instructional Conversations." *arXiv preprint arXiv:2305.14233*, 2023.

[Dee24] DeepSeek-AI. "DeepSeek-V3 Technical Report." *CoRR*, **abs/2412.19437**, 2024.

[DGC24] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vázquez, Nicolas Chapados, and Alexandre Lacoste. "WorkArena: How Capable are Web Agents at Solving Common Knowledge Work Tasks?" In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 11642–11662. PMLR / OpenReview.net, 2024.

[DJP24] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. "The llama 3 herd of models." *arXiv preprint arXiv:2407.21783*, 2024.

[DPW24] Hanxing Ding, Liang Pang, Zihao Wei, Huawei Shen, and Xueqi Cheng. "Retrieve Only When It Needs: Adaptive Retrieval Augmentation for Hallucination Mitigation in Large Language Models." *CoRR*, **abs/2402.10612**, 2024.

[DRS19] Emily Dinan, Stephen Roller, Kurt Shuster, Angela Fan, Michael Auli, and Jason Weston. "Wizard of Wikipedia: Knowledge-Powered Conversational Agents." In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[DWA23] Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, et al. "CrossCodeEval: A Diverse and Multilingual Benchmark for Cross-File Code Completion." In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[DWA24] Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. "Co-CoMIC: Code Completion by Jointly Modeling In-file and Cross-file Context." In *LREC/COLING*, pp. 3433–3445, 2024.

[DWR22] Andrew Drozdov, Shufan Wang, Razieh Rahimi, Andrew McCallum, Hamed Zamani, and Mohit Iyyer. "You can't pick your neighbors, or can you? When and How to Rely on Retrieval in the kNN-LM." In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 2997–3007, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[DWZ24] Yiming Du, Hongru Wang, Zhengyi Zhao, Bin Liang, Baojun Wang, Wanjun Zhong, Zezhong Wang, and Kam-Fai Wong. "PerLTQA: A Personal Long-Term Memory Dataset for Memory Classification, Retrieval, and Fusion in Question Answering." In Kam-Fai Wong, Min Zhang, Ruifeng Xu, Jing Li, Zhongyu Wei, Lin Gui, Bin Liang, and Runcong Zhao, editors, *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pp. 152–164, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[DZY26] Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. "CL-bench: A Benchmark for Context Learning." *CoRR*, **abs/2602.03587**, 2026.

[Eng26] Engram. "Understanding a Law Firm through Study." `https://engram.com/blog/legal-agents-with-memory`, August 2026. In partnership with Harvey. Accessed 2026-10-11.

[EOF12] Miles Efron, Peter Organisciak, and Katrina Fenlon. "Improving retrieval of short texts through document expansion." In William R. Hersh, Jamie Callan, Yoelle Maarek, and Mark Sanderson, editors, *The 35th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '12, Portland, OR, USA, August 12-16, 2012*, pp. 911–920. ACM, 2012.

[FIJ26] Gaodan Fang, Vatche Isahagian, K. R. Jayaram, Ritesh Kumar, Vinod Muthusamy, Punleuk Oum, and Gegi Thomas. "Trajectory-Informed Memory Generation for Self-Improving Agent Systems." *CoRR*, **abs/2603.10600**, 2026.

[FLW26] Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. "Memp: Exploring Agent Procedural Memory." In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, San Diego, California, United States, July 2026. Association for Computational Linguistics.

[FMM24] Jinyuan Fang, Zaiqiao Meng, and Craig MacDonald. "TRACE the Evidence: Constructing Knowledge-Grounded Reasoning Chains for Retrieval-Augmented Generation." In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8472–8494, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[FP03] Martha S. Feldman and Brian T. Pentland. "Reconceptualizing Organizational Routines as a Source of Flexibility and Change." *Administrative Science Quarterly*, **48**(1):94–118, March 2003.

[FPN24] Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. "Data Engineering for Scaling Language Models to 128K Context." In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[FWL26] Shicheng Fang, Yuxin Wang, Xiaoran Liu, Jiahao Lu, Chuanyuan Tan, Xinchi Chen, Yining Zheng, Xuanjing Huang, and Xipeng Qiu. "AgentLongBench: A Controllable Long Benchmark For Long-Contexts Agents via Environment Rollouts." *CoRR*, **abs/2601.20730**, 2026.

[GJW24] Melody Y. Guan, Manas Joglekar, Eric Wallace, Saachi Jain, Boaz Barak, Alec Helyar, Rachel Dias, Andrea Vallone, Hongyu Ren, Jason Wei, et al. "Deliberative Alignment: Reasoning Enables Safer Language Models." *CoRR*, **abs/2412.16339**, 2024.

[GLD22] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. "UniXcoder: Unified Cross-Modal Pre-training for Code Representation." In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7212–7225, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[GLT20] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. "Retrieval Augmented Language Model Pre-Training." In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3929–3938. PMLR, 13–18 Jul 2020.

[GMK25] Etash Kumar Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. "OpenThoughts: Data Recipes for Reasoning Models." *CoRR*, **abs/2506.04178**, 2025.

[GNH24] Zitian Gao, Boye Niu, Xuzheng He, Haotian Xu, Hongzhang Liu, Aiwei Liu, Xuming Hu, and Lijie Wen. "Interpretable Contrastive Monte Carlo Tree Search Reasoning." *CoRR*, **abs/2410.01707**, 2024.

[Goo25] Google DeepMind. "Gemini 3 Pro Model Card." `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf`, 2025. Accessed: 2026-04-23.

[Goo26a] Google DeepMind. "Gemini 3.1 Pro Model Card.", February 2026. Published February 19, 2026.

[Goo26b] Google DeepMind. "Gemini 3.7 Flash Model Card.", August 2026. Published August 13, 2026.

[GPP26] Niko Grupen, Gabe Pereyra, and Julio Pereyra. "Open-Sourcing Harvey's Long Horizon Legal Agent Benchmark." `https://www.harvey.ai/blog/introducing-harveys-legal-agent-benchmark`, May 2026. Harvey.

[GSG24a] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. "CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing." In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[GSG24b] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. "HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models." *CoRR*, **abs/2405.14831**, 2024.

[GYZ25] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. "DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning." *Nature*, **645**(8081):633–638, September 2025.

[HBK21] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. "Measuring Mathematical Problem Solving With the MATH Dataset." In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

[HCC25] Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. "HiAgent: Hierarchical Working Memory Management for Solving Long-Horizon Agent Tasks with Large Language Model." In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*

(Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025, pp. 32779–32798. Association for Computational Linguistics, 2025.

[HD17] Vincent J. Hellendoorn and Premkumar Devanbu. "Are deep neural networks the best choice for modeling source code?" In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE'17, pp. 763–773. ACM, August 2017.

[HNB21] Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. "Efficient Nearest Neighbor Language Models." In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5703–5714, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[HR04] Rosco Hill and Joe Rideout. "Automatic Method Completion." In *19th IEEE International Conference on Automated Software Engineering (ASE 2004), 20-25 September 2004, Linz, Austria*, pp. 228–235. IEEE Computer Society, 2004.

[HSA25] Michael Hassid, Gabriel Synnaeve, Yossi Adi, and Roy Schwartz. "Don't Overthink it. Preferring Shorter Thinking Chains for Improved LLM Reasoning." *CoRR*, **abs/2505.17813**, 2025.

[HSK24] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. "RULER: What's the Real Context Size of Your Long-Context Language Models?" *CoRR*, **abs/2404.06654**, 2024.

[HWZ26] Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, et al. "MemoryArena: Benchmarking Agent Memory in Interdependent Multi-Session Agentic Tasks." *CoRR*, **abs/2602.16313**, 2026.

[HZC24] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. "MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework." In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[Jac12] Paul Jaccard. "The distribution of the flora in the alpine zone. 1." *New phytologist*, **11**(2):37–50, 1912.

[JBC24] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. "Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity." In Kevin Duh, Helena Gomez, and Steven

Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7036–7050, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[JCW17] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. "TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension." In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[JDL19] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. "PubMedQA: A Dataset for Biomedical Research Question Answering." In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2567–2577, Hong Kong, China, November 2019. Association for Computational Linguistics.

[JHC25] Bowen Jiang, Zhuoqun Hao, Young-Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle H. Ungar, Camillo J. Taylor, and Dan Roth. "Know Me, Respond to Me: Benchmarking LLMs for Dynamic User Profiling and Personalized Responses at Scale." *CoRR*, **abs/2504.14225**, 2025.

[JHG25] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code." In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[JWL23] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. "LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models." In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 13358–13376. Association for Computational Linguistics, 2023.

[JXG23] Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. "Active Retrieval Augmented Generation." In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7969–7992, Singapore, December 2023. Association for Computational Linguistics.

[JYS25] Bowen Jiang, Yuan Yuan, Maohao Shen, Zhuoqun Hao, Zhangchen Xu, Zichen Chen, Ziyi Liu, Anvesh Rao Vijjini, Jiashu He, Hanchao Yu, et al. "PersonaMem-v2: Towards Personalized Intelligence via Learning Implicit User Personas and Agentic Memory." *CoRR*, **abs/2512.06688**, 2025.

[JZA23] Nihal Jain, Dejiao Zhang, Wasi Uddin Ahmad, Zijian Wang, Feng Nan, Xiaopeng Li, Ming Tan, Ramesh Nallapati, Baishakhi Ray, Parminder Bhatia, et al. "ContraCLM: Contrastive Learning For Causal Language Model." In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6436–6459, Toronto, Canada, July 2023. Association for Computational Linguistics.

[Kam23] Gregory Kamradt. "Needle In A Haystack - Pressure Testing LLMs." GitHub, 2023.

[KCA22] Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. "Language models (mostly) know what they know." *ArXiv preprint*, **abs/2207.05221**, 2022.

[KCH24] Jiho Kim, Woosog Chay, Hyeonji Hwang, Daeun Kyung, Hyunseung Chung, Eunbyeol Cho, Yohan Jo, and Edward Choi. "DialSim: A Real-Time Simulator for Evaluating Long-Term Dialogue Understanding of Conversational Agents.", 2024.

[KGR22] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. "Large Language Models are Zero-Shot Reasoners." In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[KKJ23] Gangwoo Kim, Sungdong Kim, Byeongguk Jeon, Joonsuk Park, and Jaewoo Kang. "Tree of Clarifications: Answering Ambiguous Questions with Retrieval-Augmented Large Language Models." In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 996–1009, Singapore, December 2023. Association for Computational Linguistics.

[KKL20] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The efficient transformer." *arXiv preprint arXiv:2001.04451*, 2020.

[KLA22] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Mu~noz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas

Wolf, et al. “The stack: 3 tb of permissively licensed source code.” *ArXiv preprint*, **abs/2211.15533**, 2022.

[KLJ20] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. “Generalization through Memorization: Nearest Neighbor Language Models.” In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[KLZ23] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. “Efficient Memory Management for Large Language Model Serving with PagedAttention.” In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP ’23, pp. 611–626. ACM, October 2023.

[KOM20] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. “Dense Passage Retrieval for Open-Domain Question Answering.” In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics.

[KPR19] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. “Natural Questions: A Benchmark for Question Answering Research.” *Transactions of the Association for Computational Linguistics*, **7**:452–466, 2019.

[KTL24] Marzena Karpinska, Katherine Thai, Kyle Lo, Tanya Goyal, and Mohit Iyyer. “One Thousand and One Pairs: A "novel" challenge for long-context language models.” In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 17048–17085. Association for Computational Linguistics, 2024.

[LAK24] Myeonghwa Lee, Seonho An, and Min-Soo Kim. “PlanRAG: A Plan-then-Retrieval Augmented Generation for Generative Large Language Models as Decision Makers.” In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6537–6555, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[LAZ23] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. “StarCoder: may the source be with you!”, 2023.

[LDH22] Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. "ReACC: A Retrieval-Augmented Code Completion Framework." In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6227–6240, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[LDJ25] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. "Search-o1: Agentic Search-Enhanced Large Reasoning Models." In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5420–5438, Suzhou, China, November 2025. Association for Computational Linguistics.

[LDS25] Xinxi Lyu, Michael Duan, Rulin Shao, Pang Wei Koh, and Sewon Min. "Frustratingly Simple Retrieval Improves Challenging, Reasoning-Intensive Benchmarks." *CoRR*, **abs/2507.01297**, 2025.

[Lev66] Vladimir I Levenshtein et al. "Binary codes capable of correcting deletions, insertions, and reversals." In *Soviet physics doklady*, volume 10, pp. 707–710. Soviet Union, 1966.

[LFW25] Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. "Evolving Deeper LLM Thinking." *CoRR*, **abs/2501.09891**, 2025.

[LFX26] Yuetai Li, Yichen Feng, Zhangchen Xu, Zixian Ma, Kaiyuan Zheng, Fengqing Jiang, Xinghua Sun, Rulin Shao, Zichen Chen, Yue Huang, et al. "JobBench: Aligning Agent Work With Human Will." *CoRR*, **abs/2605.26329**, 2026.

[LIX23] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. "G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment." In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522, Singapore, December 2023. Association for Computational Linguistics.

[LKB24] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. "Let's Verify Step by Step." In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[LLH24] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. "Lost in the Middle: How Language Models Use Long Contexts." *Trans. Assoc. Comput. Linguistics*, **12**:157–173, 2024.

[LLL22] Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. "Generated Knowledge Prompting for Commonsense Reasoning." In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3154–3169, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[LLM26] Xiaoyuan Liu, Tian Liang, Dongyang Ma, Deyu Zhou, Haitao Mi, Pinjia He, and Yan Wang. "The Pensieve Paradigm: Stateful Language Models Mastering Their Own Context." *CoRR*, **abs/2602.12108**, 2026.

[LPP20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020.

[LPZ24] Yanming Liu, Xinyue Peng, Xuhong Zhang, Weihao Liu, Jianwei Yin, Jiannan Cao, and Tianyu Du. "RA-ISF: Learning to Answer and Understand from Retrieval Augmentation via Iterative Self-Feedback." In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4730–4749, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[LSW25] Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E. Gonzalez. "Sleep-time Compute: Beyond Inference Scaling at Test-time." *CoRR*, **abs/2504.13171**, 2025.

[LTH26] Shuai Liu, Shulin Tian, Kairui Hu, Yuhao Dong, Zhe Yang, Bo Li, Jingkang Yang, Chen Change Loy, and Ziwei Liu. "FileGram: Grounding Agent Personalization in File-System Behavioral Traces." *arXiv preprint arXiv:2604.04901*, 2026.

[LTZ23] Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jingyuan Wang, Jian-Yun Nie, and Ji-Rong Wen. "The Web Can Be Your Oyster for Improving Language Models." In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 728–746, Toronto, Canada, July 2023. Association for Computational Linguistics.

[LY25] Philip Lippmann and Jie Yang. "Style over Substance: Distilled Language Models Reason Via Stylistic Replication." *CoRR*, **abs/2504.01738**, 2025.

[LYS23] Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. "Think-in-Memory: Recalling and Post-thinking Enable LLMs with Long-Term Memory." *CoRR*, **abs/2311.08719**, 2023.

[LYZ24] Hao Li, Chenghao Yang, An Zhang, Yang Deng, Xiang Wang, and Tat-Seng Chua. "Hello Again! LLM-powered Personalized Agent for Long-term Dialogue." *CoRR*, **abs/2406.05925**, 2024.

[LZL26] Xinze Li, Ziyue Zhu, Siyuan Liu, Yubo Ma, Yuhang Zang, Yixin Cao, and Aixin Sun. "EMemBench: Interactive Benchmarking of Episodic Memory for VLM Agents." *CoRR*, **abs/2601.16690**, 2026.

[LZX26] Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, et al. "SkillNet: Create, Evaluate, and Connect AI Skills." *CoRR*, **abs/2603.04448**, 2026.

[Mat24] Mathematical Association of America. "American Invitational Mathematics Examination (AIME)." `https://maa.org/math-competitions/american-invitational-mathematics-examination-aime`, February 2024. American Invitational Mathematics Examination (AIME) 2024.

[MAZ23] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. "When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories." In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9802–9822, Toronto, Canada, July 2023. Association for Computational Linguistics.

[MCK18] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. "Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering." In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[MDD25] Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. "NoLiMa: Long-Context Evaluation Beyond Literal Matching." In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025.

[MGA25] Charlie Masters, Marta Grzeskiewicz, and Stefano V. Albrecht. "ARCANE: A Multi-Agent Framework for Interpretable and Configurable Alignment." *CoRR*, **abs/2512.06196**, 2025.

[MHH24] Tong Mu, Alec Helyar, Johannes Heidecke, Joshua Achiam, Andrea Vallone, Ian D Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. "Rule Based Rewards for Language Model Safety." In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[Mic23] Microsoft. "Announcing Microsoft Copilot, Your Everyday AI Companion.", 2023. Accessed: September 15, 2024.

[MLG23] Jesse Mu, Xiang Li, and Noah D. Goodman. "Learning to Compress Prompts with Gist Tokens." In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[MLT24] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. "Evaluating Very Long-Term Conversational Memory of LLM Agents." In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[MLZ25] Yutao Mou, Yuxiao Luo, Shikun Zhang, and Wei Ye. "SaRO: Enhancing LLM Safety through Reasoning-based Alignment." *arXiv preprint arXiv:2504.09420*, 2025.

[MMS25] Viktor Moskvoretskii, Maria Marina, Mikhail Salnikov, Nikolay Ivanov, Sergey Pletenev, Daria Galimzianova, Nikita Krayko, Vasily Konovalov, Irina Nikishina, and Alexander Panchenko. "Adaptive Retrieval Without Self-Knowledge? Bringing Uncertainty Back Home." In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6355–6384, Vienna, Austria, July 2025. Association for Computational Linguistics.

[MTG23] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. "Self-Refine: Iterative Refinement with Self-Feedback." In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[MTM23] Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. "MTEB: Massive Text Embedding Benchmark." In Andreas Vlachos and Isabelle Augenstein, editors, *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2014–2037, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.

[MYS25] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. "s1: Simple test-time scaling." In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics.

[MZA23] Todor Markov, Chong Zhang, Sandhini Agarwal, Florentine Eloundou Nekoul, Theodore Lee, Steven Adler, Angela Jiang, and Lilian Weng. "A Holistic Approach to Undesired Content Detection in the Real World." In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 15009–15018. AAAI Press, 2023.

[Nat] National Center for Biotechnology Information (NCBI). "PubMed." `https://pubmed.ncbi.nlm.nih.gov/`. U.S. National Library of Medicine (NLM), National Institutes of Health (NIH).

[Neo26] NeoCognition Team. "ApprenticeBench." `https://apprenticebench.com/`, September 2026. Accessed 2026-10-11.

[NGM25] Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, Jian Zhang, and Jane Polak Scowcroft. "Nemotron-Post-Training-Dataset-v1.", July 2025.

[NHX23] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. "CodeGen2: Lessons for Training LLMs on Programming and Natural Languages.", 2023.

[NPH23] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis." In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[Ope] OpenAI. "Codex CLI." `https://github.com/openai/codex`. Open-source coding agent software. Accessed: 2026-04-24.

[Ope22] OpenAI. "ChatGPT.", 2022. Accessed: September 15, 2024.

[Ope23a] OpenAI. "GPT-4 Technical Report." arXiv:2303.08774, 2023.

[Ope23b] OpenAI. "Using GPT-4 for Content Moderation." https://openai.com/index/using-gpt-4-for-content-moderation/, August 2023. Accessed 2025-07-04.

[Ope24a] OpenAI. "Memory and New Controls for ChatGPT." https://openai.com/index/memory-and-new-controls-for-chatgpt/, 2024. Accessed: September 15, 2024.

[Ope24b] OpenAI. "OpenAI o1 System Card and Learning to Reason with LLMs." https://arxiv.org/abs/2412.16720 and https://openai.com/index/learning-to-reason-with-llms, 2024. Technical reports accompanying the OpenAI o1 and o1-mini reasoning models.

[Ope25a] OpenAI. "Introducing GPT-5.2." https://openai.com/index/introducing-gpt-5-2/, December 2025. Accessed: 2025-12-21.

[Ope25b] OpenAI. "Update to GPT-5 System Card: GPT-5.2." https://openai.com/index/gpt-5-system-card-update-gpt-5-2/, 2025. Accessed: 2026-04-23.

[Ope26] OpenAI. "GPT-5 System Card.", 2026.

[OYC26] Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, et al. "SkillOS: Learning Skill Curation for Self-Evolving Agents." *CoRR*, **abs/2605.06614**, 2026.

[OYH25] Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, et al. "ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory." *CoRR*, **abs/2509.25140**, 2025.

[OZX25] Siru Ouyang, Xinyu Zhu, Zilin Xiao, Minhao Jiang, Yu Meng, and Jiawei Han. "RAST: Reasoning Activation in LLMs via Small-model Transfer." *CoRR*, **abs/2506.15710**, 2025.

[Par72] D. L. Parnas. "On the criteria to be used in decomposing systems into modules." *Communications of the ACM*, **15**(12):1053–1058, December 1972.

[PDP25] Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, et al. "GDPval: Evaluating AI Model Performance on Real-World Economically Valuable Tasks." *CoRR*, **abs/2510.04374**, 2025.

[PFP23] Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. "MemGPT: Towards LLMs as Operating Systems." *CoRR*, **abs/2310.08560**, 2023.

[PLG25] Sungjin Park, Xiao Liu, Yeyun Gong, and Edward Choi. "Ensembling Large Language Models with Process Reward-Guided Tree Search for Better Complex Reasoning." In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10256–10277, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

[PPR25] Tanmay Parekh, Pradyot Prakash, Alexander Radovic, Akshay Shekher, and Denis Savenkov. "Dynamic Strategy Planning for Efficient Question Answering with Large Language Models." In *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025.

[PZL23] Hengzhi Pei, Jinman Zhao, Leonard Lausen, Sheng Zha, and George Karypis. "Better Context Makes Better Code Language Models: A Case Study on Function Call Argument Completion." In *AAAI*, pp. 5230–5238, 2023.

[QSL25] Yuxiao Qu, Anikait Singh, Yoonho Lee, Amrith Setlur, Ruslan Salakhutdinov, Chelsea Finn, and Aviral Kumar. "RLAD: Training LLMs to Discover Abstractions for Solving Reasoning Problems." *CoRR*, **abs/2510.02263**, 2025.

[Qwe24] Qwen Team. "QwQ: A Family of Open Reasoning Models." `https://qwenlm.github.io/blog/qwq-32b-preview/`, 2024. Technical report and model card for the QwQ reasoning model family.

[Qwe26a] Qwen Team. "Qwen3.5: Towards Native Multimodal Agents.", February 2026.

[Qwe26b] Qwen Team. "Qwen3.6-27B: Flagship-Level Coding in a 27B Dense Model.", April 2026. Official Qwen release report.

[RCM19] Siva Reddy, Danqi Chen, and Christopher D. Manning. "CoQA: A Conversational Question Answering Challenge." *Transactions of the Association for Computational Linguistics*, **7**:249–266, 2019.

[RGG23] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. "Code llama: Open foundation models for code." *ArXiv preprint*, **abs/2308.12950**, 2023.

[RHS24] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. "GPQA: A Graduate-Level Google-Proof Q&A Benchmark." In *First Conference on Language Modeling*, 2024.

[RLD23] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. "In-Context Retrieval-Augmented Language Models." *Transactions of the Association for Computational Linguistics*, **11**:1316–1331, 2023.

[RSM23] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. "Direct Preference Optimization: Your Language Model is Secretly a Reward Model." In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[SAT24] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. "RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval." In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[SCG23] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. "Reflexion: language agents with verbal reinforcement learning." In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[SCM23] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. "Large Language Models Can Be Easily Distracted by Irrelevant Context." In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31210–31227. PMLR, 2023.

[SDD23] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. "Toolformer: Language Models Can Teach Themselves to Use Tools." In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[SDF20] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. "IntelliCode compose: code generation using transformer." In *Proceedings of the 28th*

ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE '20, pp. 1433–1443. ACM, November 2020.

[SGS23] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. "Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy." In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9248–9274, Singapore, December 2023. Association for Computational Linguistics.

[SJM25] Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. "R1-Searcher: Incentivizing the Search Capability in LLMs via Reinforcement Learning." *CoRR*, **abs/2503.05592**, 2025.

[SKV23] Disha Shrivastava, Denis Kocetkov, Harm de Vries, Dzmitry Bahdanau, and Torsten Scholak. "RepoFusion: Training Code Models to Understand Your Repository." *ArXiv preprint*, **abs/2306.10998**, 2023.

[SLD22] Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. "ASQA: Factoid Questions Meet Long-Form Answers." In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 8273–8288, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[SLT23] Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. "Repository-Level Prompt Generation for Large Language Models of Code." In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31693–31715. PMLR, 2023.

[SLX25] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. "Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning." In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

[SMY24] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. "REPLUG: Retrieval-Augmented Black-Box Language Models." In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8371–8384, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[SQK25] Rulin Shao, Rui Qiao, Varsha Kishore, Niklas Muennighoff, Xi Victoria Lin, Daniela Rus, Bryan Kian Hsiang Low, Sewon Min, Wen-tau Yih, Pang Wei Koh, et al. "ReasonIR: Training Retrievers for Reasoning Tasks." *CoRR*, **abs/2504.20595**, 2025.

[STA24] Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. "DRAGIN: Dynamic Retrieval Augmented Generation based on the Real-time Information Needs of Large Language Models." In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12991–13013, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[SWL20] Vered Shwartz, Peter West, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. "Unsupervised Commonsense Question Answering with Self-Talk." In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4615–4629, Online, November 2020. Association for Computational Linguistics.

[SYB25] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. "Dynamic Cheatsheet: Test-Time Learning with Adaptive Memory." *CoRR*, **abs/2504.07952**, 2025.

[THY25] Xiangru Tang, Tianyu Hu, Muyang Ye, Yanjun Shao, Xunjian Yin, Siru Ouyang, Wangchunshu Zhou, Pan Lu, Zhuosheng Zhang, Yilun Zhao, et al. "ChemAgent: Self-updating Library in Large Language Models Improves Chemical Reasoning." *CoRR*, **abs/2501.06590**, 2025.

[THZ26] CocoaBench Team, Shibo Hao, Zhining Zhang, Zhiqi Liang, Tianyang Liu, Yuheng Zha, Qiyue Gao, Jixuan Chen, Zilong Wang, Zhoujun Cheng, et al. "CocoaBench: Evaluating Unified Digital Agents in the Wild.", 2026.

[TMS23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. "Llama 2: Open foundation and fine-tuned chat models." *ArXiv preprint*, **abs/2307.09288**, 2023.

[TSD14] Zhaopeng Tu, Zhendong Su, and Premkumar T. Devanbu. "On the localness of software." In Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey, editors, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pp. 269–280. ACM, 2014.

[TSY25] Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J. Ross Mitchell. "Beyond a Million Tokens: Benchmarking and Enhancing Long-Term Memory in LLMs." *CoRR*, **abs/2510.27246**, 2025.

[TVC18] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. "FEVER: a Large-scale Dataset for Fact Extraction and VERification." In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 809–819, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[TWM06] Tao Tao, Xuanhui Wang, Qiaozhu Mei, and ChengXiang Zhai. "Language Model Information Retrieval with Document Expansion." In Robert C. Moore, Jeff Bilmes, Jennifer Chu-Carroll, and Mark Sanderson, editors, *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pp. 407–414, New York City, USA, June 2006. Association for Computational Linguistics.

[VD20] Bertie Vidgen and Leon Derczynski. "Directions in abusive language training data, a systematic review: Garbage in, garbage out." *Plos one*, **15**(12):e0243300, 2020.

[VMF26] Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, et al. "APEX-Agents." *CoRR*, **abs/2601.14242**, 2026.

[WAZ24] Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. "Repoformer: Selective Retrieval for Repository-Level Code Completion." In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 53270–53290. PMLR, 21–27 Jul 2024.

[WCB14] Jason Weston, Sumit Chopra, and Antoine Bordes. "Memory networks." *arXiv preprint arXiv:1410.3916*, 2014.

[WCZ24] Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. "OfficeBench: Benchmarking Language Agents across Multiple Applications for Office Automation." *CoRR*, **abs/2407.19056**, 2024.

[WDC23] Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. "Augmenting Language Models with Long-Term Memory." In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[WGC24] Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, et al. "MEMORYLLM: Towards

Self-Updatable Large Language Models." In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 50453–50466. PMLR / OpenReview.net, 2024.

[WGC26] Di Wu, Jia-Chen Gu, Kai-Wei Chang, and Nanyun Peng. "Self-Routing RAG: Binding Selective Retrieval with Knowledge Verbalization." In *Findings of the Association for Computational Linguistics: AACL 2026*, 2026.

[WGY24] Di Wu, Jia-Chen Gu, Fan Yin, Nanyun Peng, and Kai-Wei Chang. "Synchronous Faithfulness Monitoring for Trustworthy Retrieval-Augmented Generation." In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9390–9406, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[WHC26] Di Wu, Rujun Han, Yanfei Chen, Yifei Ming, Han Yu, Yufan Zhuang, Tomas Pfister, and Chen-Yu Lee. "ReOrg: Evolving Organizational Reflection Skills for Digital Agents." Manuscript. `https://openreview.net/forum?id=aRQEiC7qgJ`, 2026.

[WJK26] Di Wu, Zixiang Ji, Asmi Kawatkar, Bryan Kwan, Jia-Chen Gu, Nanyun Peng, and Kai-Wei Chang. "LongMemEval-V2: Evaluating Long-Term Agent Memory Toward Experienced Colleagues." arXiv:2605.12493, 2026.

[WLD18] Wei Wei, Quoc Le, Andrew Dai, and Jia Li. "AirDialogue: An Environment for Goal-Oriented Dialogue Research." In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3844–3854, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[WLJ26] Di Wu, Siyue Liu, Zixiang Ji, Ya-Liang Chang, Zhe-Yu Liu, Andrew Pleffer, and Kai-Wei Chang. "Open-Domain Safety Policy Construction." In Vera Demberg, Kentaro Inui, and Lluís Marquez, editors, *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4981–4999, Rabat, Morocco, March 2026. Association for Computational Linguistics.

[WLL24] Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. "RAT: Retrieval Augmented Thoughts Elicit Context-Aware Reasoning in Long-Horizon Generation." *CoRR*, **abs/2403.05313**, 2024.

[WLS23] Yile Wang, Peng Li, Maosong Sun, and Yang Liu. "Self-Knowledge Guided Retrieval Augmentation for Large Language Models." In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational*

Linguistics: EMNLP 2023, pp. 10303–10315, Singapore, December 2023. Association for Computational Linguistics.

[WLS24] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. "Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations." In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[WMF25] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. "Agent Workflow Memory." In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 13–19 Jul 2025.

[WRH22] Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. "Memorizing transformers." *arXiv preprint arXiv:2203.08913*, 2022.

[WSC24] Di Wu, Xiaoxian Shen, and Kai-Wei Chang. "MetaKP: On-Demand Keyphrase Generation." In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8420–8437, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[WSD21] Yanlin Wang, Ensheng Shi, Lun Du, Xiaodi Yang, Yuxuan Hu, Shi Han, Hongyu Zhang, and Dongmei Zhang. "Cocosum: Contextual code summarization with multi-relational graph neural network." *ArXiv preprint*, **abs/2107.01933**, 2021.

[WSY26] Di Wu, Devendra Singh Sachan, Wen-tau Yih, and Mingda Chen. "Procedural Knowledge at Scale Improves Reasoning." In *Conference on Language Modeling (COLM)*, 2026.

[WTL25] Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian J. McAuley, and Xiaojian Wu. "Mem-α: Learning Memory Construction via Reinforcement Learning." *CoRR*, **abs/2509.25911**, 2025.

[WWC26] Di Wu, Yixin Wan, and Kai-Wei Chang. "VisRet: Visualization Improves Knowledge-Intensive Text-to-Image Retrieval." In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25982–26001, San Diego, California, United States, July 2026. Association for Computational Linguistics.

[WWS22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[WWS23] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. "Self-Consistency Improves Chain of Thought Reasoning in Language Models." In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[WWY25] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. "LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory." In *The Thirteenth International Conference on Learning Representations*, 2025.

[WXJ24] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. "Voyager: An Open-Ended Embodied Agent with Large Language Models." *Trans. Mach. Learn. Res.*, **2024**, 2024.

[WYC24] Di Wu, Da Yin, and Kai-Wei Chang. "KPEval: Towards Fine-Grained Semantic-Based Keyphrase Evaluation." In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1959–1981, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[WZY25] Wenyi Wu, Kun Zhou, Ruoxin Yuan, Vivian Yu, Stephen Wang, Zhiting Hu, and Biwei Huang. "Auto-scaling Continuous Memory for GUI Agent." *CoRR*, **abs/2510.09038**, 2025.

[xAI25] xAI. "Grok 4.1 Model Card." `https://data.x.ai/2025-11-17-grok-4-1-model-card.pdf`, 2025. Accessed: 2026-04-23.

[XDW25] Yuhui Xu, Hanze Dong, Lei Wang, Doyen Sahoo, Junnan Li, and Caiming Xiong. "Scalable Chain of Thoughts via Elastic Reasoning." *CoRR*, **abs/2505.05315**, 2025.

[XGD23] Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. "Baize: An Open-Source Chat Model with Parameter-Efficient Tuning on Self-Chat Data." In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6268–6278, Singapore, December 2023. Association for Computational Linguistics.

[XGW22] Xinchao Xu, Zhibin Gou, Wenquan Wu, Zheng-Yu Niu, Hua Wu, Haifeng Wang, and Shihang Wang. "Long Time No See! Open-Domain Conversation with Long-Term Persona Memory." In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2639–2650, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[XJL24] Guangzhi Xiong, Qiao Jin, Zhiyong Lu, and Aidong Zhang. "Benchmarking Retrieval-Augmented Generation for Medicine." In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 6233–6251, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[XLM25] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. "A-MEM: Agentic Memory for LLM Agents." *CoRR*, **abs/2502.12110**, 2025.

[XSC24] Fangyuan Xu, Weijia Shi, and Eunsol Choi. "RECOMP: Improving Retrieval-Augmented LMs with Context Compression and Selective Augmentation." In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[XSL25] Frank (Fangzheng) Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. "TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks." In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025.

[XSW22] Jing Xu, Arthur Szlam, and Jason Weston. "Beyond Goldfish Memory: Long-Term Open-Domain Conversation." In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5180–5197, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[YF02] Yunwen Ye and Gerhard Fischer. "Supporting reuse by delivering task-relevant and personalized information." In *Proceedings of the 24th international conference on Software engineering - ICSE '02*, ICSE '02, p. 513. ACM Press, 2002.

[YGZ24] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. "Corrective Retrieval Augmented Generation." *CoRR*, **abs/2401.15884**, 2024.

[YIW23] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. "Generate rather than Retrieve: Large Language Models are Strong Context Generators." In *The*

Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023.

[YLY25] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. "Qwen3 Technical Report." *CoRR*, **abs/2505.09388**, 2025.

[YQP25] Zijun Yao, Weijian Qi, Liangming Pan, Shulin Cao, Linmei Hu, Liu Weichuan, Lei Hou, and Juanzi Li. "SeaKR: Self-aware Knowledge Retrieval for Adaptive Retrieval Augmented Generation." In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27022–27043, Vienna, Austria, July 2025. Association for Computational Linguistics.

[YSC24] Fan Yin, Jayanth Srinivasa, and Kai-Wei Chang. "Characterizing Truthfulness in Large Language Model Generations with Local Intrinsic Dimension." In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[YVL23] Fan Yin, Jesse Vig, Philippe Laban, Shafiq Joty, Caiming Xiong, and Chien-Sheng Wu. "Did You Read the Instructions? Rethinking the Effectiveness of Task Definitions in Instruction Learning." In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3063–3079, Toronto, Canada, July 2023. Association for Computational Linguistics.

[YYC25] Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. "ReasonFlux: Hierarchical LLM Reasoning via Scaling Thought Templates." *CoRR*, **abs/2502.06772**, 2025.

[YYH25] Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Hinrich Schütze, Volker Tresp, and Yunpu Ma. "Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize Memories via Reinforcement Learning." *CoRR*, **abs/2508.19828**, 2025.

[YYZ23] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[YYZ24a] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. "Qwen2. 5 technical report." *ArXiv preprint*, **abs/2412.15115**, 2024.

[YYZ24b] Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. "Buffer of Thoughts: Thought-Augmented Reasoning with Large Language Models." In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[YZ21] Wenjie Yin and Arkaitz Zubiaga. "Towards generalisable hate speech detection: a review on obstacles and solutions." *PeerJ Computer Science*, **7**:e598, 2021.

[YZP23] Wenhao Yu, Hongming Zhang, Xiaoman Pan, Kaixin Ma, Hongwei Wang, and Dong Yu. "Chain-of-note: Enhancing robustness in retrieval-augmented language models." *arXiv preprint arXiv:2311.09210*, 2023.

[YZY23] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. "ReAct: Synergizing Reasoning and Acting in Language Models." In *International Conference on Learning Representations (ICLR)*, 2023.

[ZAX23] Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. "DocPrompting: Generating Code by Retrieving the Docs." In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[ZC21] Michael Zhang and Eunsol Choi. "SituatedQA: Incorporating Extra-Linguistic Contexts into QA." In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7371–7387, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[ZCL22] Daoguang Zan, Bei Chen, Zeqi Lin, Bei Guan, Wang Yongji, and Jian-Guang Lou. "When Language Model Meets Private Library." In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 277–288, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[ZCS23] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, et al. "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena.", 2023.

[ZCZ23] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. "RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation." In Houda

Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2471–2484, Singapore, December 2023. Association for Computational Linguistics.

[ZGG24] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. "MemoryBank: Enhancing Large Language Models with Long-Term Memory." In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pp. 19724–19731. AAAI Press, 2024.

[Zha23] Dun Zhang. "STELLA EN 1.5B v5." `https://huggingface.co/dunzhang/stella_en_1.5B_v5`, 2023. Accessed: September 15, 2024.

[ZHX24] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. "ExpeL: LLM Agents Are Experiential Learners." In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pp. 19632–19642. AAAI Press, 2024.

[ZHX26] Xinping Zhao, Xinshuo Hu, Jiaxin Xu, Danyu Tang, Xin Zhang, Mengjia Zhou, Yan Zhong, Yao Zhou, Zifei Shan, Meishan Zhang, et al. "LMEB: Long-horizon Memory Embedding Benchmark." *CoRR*, **abs/2603.12572**, 2026.

[ZKR20] Eric Zeng, Tadayoshi Kohno, and Franziska Roesner. "Bad News: Clickbait and Deceptive Ads on News and Misinformation Websites." In *Workshop on Technology and Consumer Protection (ConPro)*, 2020.

[ZLB26] Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. "MemSkill: Learning and Evolving Memory Skills for Self-Evolving Agents." *CoRR*, **abs/2602.02474**, 2026.

[ZLC22] Zexuan Zhong, Tao Lei, and Danqi Chen. "Training Language Models with Memory Augmentation." In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5657–5673, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[ZLC23] Tianhua Zhang, Hongyin Luo, Yung-Sung Chuang, Wei Fang, Luc Gaitskell, Thomas Hartvigsen, Xixin Wu, Danny Fox, Helen Meng, and James R. Glass. "Interpretable Unified Language Checking." *CoRR*, **abs/2304.03728**, 2023.

[ZPR24] Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. "SELF-DISCOVER: Large Language Models Self-Compose Reasoning Structures." In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[ZPW24] Hongming Zhang, Xiaoman Pan, Hongwei Wang, Kaixin Ma, Wenhao Yu, and Dong Yu. "Cognitive Kernel: An Open-source Agent System towards Generalist Autopilots.", 2024.

[ZXZ24] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. "WebArena: A Realistic Web Environment for Building Autonomous Agents." In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[ZYH26] Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, et al. "AMA-Bench: Evaluating Long-Horizon Memory for Agentic Applications." *CoRR*, **abs/2602.22769**, 2026.